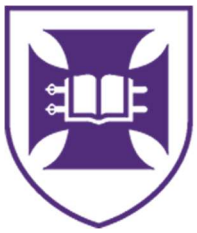


THE UNIVERSITY OF QUEENSLAND
AUSTRALIA

Enhancing Electrical Consumption Forecasts Using Bayesian Optimisation and Genetic Algorithms

by

David Gaul

School of Electrical Engineering and Computer Science

The University of Queensland.

Submitted for the degree of Bachelor of Engineering (Honours) in the division of Electrical and Computer Engineering.

Date of Submission

6/11/23

Declaration of Ownership

David Gaul

d.gaul@uqconnect.edu.au

6/11/23

Prof Michael Brünig,

Head of School

School of Electrical Engineering and Computer Science

The University of Queensland

St Lucia QLD 4072

Dear Professor Brünig,

In accordance with the requirements of the Degree of Bachelor of Engineering (Honours) in the School of Electrical Engineering and Computer Science, I submit the following thesis entitled

"Enhancing Electrical Consumption Forecasts using Bayesian Optimisation and Genetic Algorithms"

The thesis was performed under the supervision of Dr Hui Ma. I declare that the work submitted in the thesis is my own, except as acknowledged in the text and footnotes, and that it has not previously been submitted for a degree at the University of Queensland or any other institution.

Yours sincerely

Electronically signed: David Gaul

Acknowledgements

It never really struck me how much work went into a thesis before I had to write my own. That was before 8 months, 14 hours in meetings, reading 38 scientific journals, writing 86 pages in reports, 6000 lines of code and countless hours of wondering why my brain isn't braining. I now know that it takes A LOT of work to write a thesis. With that in mind, there's three people I would like to thank for being with me during this experience.

Thank you to Hui Ma, my university supervisor. Our first meeting wasn't at the start of the thesis; it came about six months before, when Xin and Vincent recommended I study under you. They both said you were their favourite supervisor on the campus, and I now see why. It's been a year since then, and you've been an endless fountain of advice and opportunity. From tutoring to applying for scholarships, exchange programs and thesis presentations… I would never have taken any of those steps if it weren't for your encouragement. Thanks for not just overseeing my good work, but ensuring I was always working to be the best I could be. If it weren't for you, I wouldn't have taken myself half as far as I did.

Thank you to Matt Amos, my CSIRO industry thesis supervisor. When you agreed to take me on, you were only one week back to work after having your first child. You'd been appointed a new role at work that came with a load of extra responsibility, and we both knew that sleep would be a rare luxury for you over the next year. To even think of taking me, someone you'd met 1 hour before, under your wing speaks levels about your character. I think it's all the small things that really shone through and made the difference. Those small things can often go unnoticed. But hey, you're a parent now. You'll get used to that. I want to say thanks for everything, because there were so many things to be grateful for.

A final and very special thanks goes to Gunz, my lifelong friend. One night, four years ago, I asked you about electrical engineering. The very next day, you'd convinced me to sign up. It's been a long four years, and so much has happened. I'm now a lot older, a lot greyer and have aged horribly. Thanks. That's because of your advice. You've been my best friend and my worst influence. I wish you were still here.

Lots of love,

David

Contributions

Bringing the thesis to this level of completion would not have been possible without valuable input from both my industry and university supervisors. As such, this section is dedicated to the numerous contributions they provided throughout the course of the entire project. The contributions that made this thesis possible are detailed below.

- MathWorks Load Prediction dataset and Webinar case study, provided by Hui Ma. This case study provided the groundworks for basic time-series forecasting of load data [1].
- GEFCOM 2012 Load data and background literature, provided by Hui Ma. The contents of the 2012 forecasting competition, as well as published literature from several key contestants was provided to provide insight into prior art [2].
- Peer-Reviewed database, provided by Hui Ma. Many of the articles used for understanding of theory and prior art were sourced from this database.
- AEMO NEM data scraper to find electricity demand data, provided by Matt Amos. Used for gathering the NEM dataset used for training and testing.
- AEMO NEM data scraper to find forecasting models, provided by Matt Amos. Used to scrape and compare AEMO's predictions from the NEM dataset.
- Proofreading and editing of thesis and project proposal, provided by Matt Amos. Vital to produce such high-quality reports.

Abstract

Time-series forecasting is the study of using historical patterns to predict future events. Appearing across a range of fields including stock markets, healthcare and electricity consumption, it is quintessential for the function of countless systems. Numerous models have been created to assist with time-series forecasting, although a recent emphasis has been placed on automated machine learning pipelines for efficient and powerful results.

In this paper, two novel methods of time-series forecasting through automated machine learning were devised. Both models were compared against a set of key performance indicators, which emphasized training speed, output accuracy and user-friendly design.

The first of these methods harnessed a hybrid Grid Search (GS) / Bayesian Optimisation (BO) model. The second model was based on genetic algorithms (GAs). A final baseline persistence model was created for comparison purposes. Each of these models were then applied to three datasets across three forecasting horizons: 1-step, 1-day and 1-week, for a total of nine trials.

At the conclusion of the results, the GA was found to have outperformed all other algorithms in eight of the nine trials. Furthermore, in 78% of trials, it finished with a mean average percent error (MAPE) of between 0.5%-2%. Its performance vastly improved over the GS/BO architecture, with an error reduction of up to 1240%. Furthermore, three of the trials was compared against the Australian Energy Market Organisation's (AEMO's) pre-dispatch forecaster for New South Wales's electricity demand. The GA was found to perform on par with AEMO in two of the three trials. For 5-minute and 1-day forecasts, AEMO outputted final predictions with 0.6% and 2.5% MAPE: outperforming the GA by 0.4% and 0.5% respectively.

The creation of the GA pipeline served to satisfy all the key performance indicators proposed by this paper. It was efficient, accurate and able to run without the use of GPUs. The inclusion of output visualisations, auto-generated Python pipeline files and customisability served to enhance user-friendliness. Furthermore, the final pipeline could be adaptable to predict on any time-series data with minimal tweaking.

This is not to say that the GA functioned perfectly. Through trialling, it was found that the GA was able to perform most accurately for forecast horizons of one day or less. Accuracy of predictions decreased significantly during week-long forecasts in two of the three datasets. This is thought to be a combination of sub-optimal feature adding and a lack of input data complexity. Enhancing the GA's long-term forecasting capabilities is among the most important recommendations for future work.

List of Tables

Table 1: Comparison of Statistical Models (Time Series) to the NN/GA implementation (ANN) proposed by Azadeh et al. (Sourced from [6]) 18
Table 2: Adjustable inputs over the data cleaning process. 35
Table 3: Adjustable hyperparameters for random forest and XGBoost models using BO. 38
Table 4: NN hyperparameters available for tuning using BO. 39
Table 5: List of genes that would be adjusted within each genome. 41
Table 6: Results for 1 step forecast on NSW electrical demand sourced from AEMO's NEM dataset. . 47
Table 7: Results for 1 day forecast on NSW electrical demand sourced from AEMO's NEM dataset. .. 47
Table 8: Results for 1 week forecast on NSW electrical demand sourced from AEMO's NEM dataset. 48
Table 9: Results for 1 step forecast on electrical consumption sourced from Mathwork's dataset. 50
Table 10: Results for 1 day forecast on electrical consumption sourced from Mathwork's dataset. ... 51
Table 11: Results for 1 week forecast on electrical consumption sourced from Mathwork's dataset. 51
Table 12: Results for 1 step forecast on electrical consumption sourced from GEFCOM's dataset. 53
Table 13: Results for 1 day forecast on electrical consumption sourced from GEFCOM's dataset..... 53
Table 14: Results for 1 week forecast on electrical consumption sourced from GEFCOM's dataset. ... 53

Table of Figures

Figure 1: Consumption for a residential consumer for (a) 8 months and (b) a three-week period. From "Forecasting Electricity Smart Meter Data using Conditional Kernel Density Estimation" by S. Arora, 2014 15
Figure 2: Summaries of density forecasts for (a) a residential consumer and (b) an SME. From "Forecasting Electricity Smart Meter Data using Conditional Kernel Density Estimation" by S. Arora, 2014 16
Figure 3: Empirical CDF comparisons of uni-formative and BO-tuned models for predictions of patient's vital signs. (a) provides a 10-15 minute prediction window, (c) 45-60 minutes and (d) 1-5 minutes (Sourced from [22]) 19
Figure 4: Instant power predictions of heatpump, EV and PV systems (Sourced from [24]) 20
Figure 5: Example of applying moving average filter to Mathwork's load forecasting dataset (Deoras, 2016) 22
Figure 6: Seasonal Decomposition of Mathwork's Household Load dataset (Deoras, 2016) 22
Figure 7: Representation of applying a sliding window of 5 entries to a time series dataset (Sourced from [28]) 23
Figure 8: Example of PCA transforming the input data onto a new set of orthogonal axes known as principal components (Sourced from [30]) 24
Figure 9: Example dendrogram created by clustering cars (Sourced from [15]) 25
Figure 10: Architecture of Neural Network (Sourced from [32]) 26
Figure 11: Architecture of LSTM Memory Cells (Sourced from [3]) 27
Figure 12: Architecture of CNNs (Sourced from [33]) 28
Figure 13: Description of BO's process of fitting black box functions (sourced from [34]) 29
Figure 14: Flowchart of Bayesian Optimisation process (Sourced from [35]) 30
Figure 15: Flowchart showing the evolutionary process employed by GAs (Sourced from [37]) 31
Figure 16: First 5 inputs of the hourly load consumption data 33
Figure 17: GEFCOM load dataset after matrix transposition 34
Figure 18: Flow chart for the BO algorithm 37
Figure 19: Online dashboard showing the performances of ML models on various datasets 40
Figure 20: Structure of the Genome Layout 41
Figure 21: Genome dictionary created for every genome within the population. 41
Figure 22: Representation of the genetic cross-over process. 43
Figure 23: Flow Chart of the GA process. 44
Figure 24: Simplified web dashboard created for GA. 45
Figure 25: Comparison of metrics for different models on AEMO's NEM dataset. 48
Figure 26: Comparison of 5-minute forecasts on AEMO's NEM dataset. 49
Figure 27: Comparison of 1-week forecasts on AEMO's NEM dataset 49
Figure 28: Metrics comparison for different models on the Mathworks dataset. 51
Figure 29: Comparison of 1-week forecasts on the Mathworks dataset. 52
Figure 30: Metrics comparison for different models on the GEFCOM dataset 53
Figure 31: Comparison of 1-week forecasts on the GEFCOM dataset. 54
Figure 32: Simplified representation of the GS/BO architecture 55
Figure 33: Example output of the GA's visualiser following 20 minutes runtime 60
Figure 34: Example output from the GS/BO results visualisation 60
Figure 35: In-built electricity consumption predictor from the GS/BO implementation 61
Figure 36: An example auto-generated .py file made from the GA's most effective genome 63

Figure 37: 1-day trend across the GEFCOM load data 64
Figure 38: Comparison of 1-week forecasts on the AEMO dataset 64
Figure 39: Comparison of 1-week forecasts on the Mathwork's dataset 65
Figure 40: Comparison of 1-step forecasts on the Mathwork's dataset 65
Figure 41: Comparison of 1-day forecasts on the Mathworks dataset 66
Figure 42: Comparison of 1-week forecasts on the GEFCOM load dataset 66
Figure 43: Comparison of 5-minute forecasts on the NEM dataset 67

1 Table of Contents

Declaration of Ownership 2
Acknowledgements 3
Contributions 4
Abstract 5
List of Tables 6
Table of Figures 7
2 Introduction 11
2.1 Project Motivation 11
2.2 Project Overview and Scope 12
2.3 Report Outline 13
3 Literature Review 14
3.1 Smart Meters 14
3.2 Feature Adding and Noise Reduction in Time Series Forecasting 14
3.3 Role of Genetic Algorithms in Optimising Forecasting Models 16
3.4 Role of Bayesian Optimisation in Time-Series Forecasting 18
3.5 Commonly used Forecasting Models within GEFCOM 2012 and 2014 20
4 Theory 21
4.1 Data usage and Feature Adding in Electrical Consumption Forecasting 21
4.1.1 Moving Average Filters (MAF) 21
4.1.2 Seasonal Decomposition 22
4.1.3 Windowing 23
4.1.4 Principal Component Analysis (PCA) 23
4.1.5 Agglomerative Clustering 25
4.2 Principles of Neural Networks 26
4.2.1 Long Short-Term Memory Networks (LSTM) 27
4.2.2 Convolutional Neural Network (CNN) 28
4.3 Optimisation Strategies 28
4.3.1 Bayesian Optimisation 29
4.3.2 Genetic Algorithms 31
5 Methodology 33
5.1.1 Dataset and Preprocessing 33
5.1.2 Data Cleaning and Feature Engineering 35
5.1.3 Performance Metrics 36
5.1.4 Bayesian Optimization Pipeline 37
5.1.5 Genetic Algorithm 41

6 Discussion and Results 47
6.1 AEMO NEM Dataset 47
6.2 Mathworks Dataset 50
6.3 GEFCOM Dataset 53
6.4 General Observations 55
7 Recommendations and Conclusion 58
7.1 Recommendations for Further Improvement 59
8 Appendices 60
9 References 68

2 Introduction

2.1 Project Motivation

Time series forecasting is a field of research that has exploded since the end of the 20th century. With ties in economics, health, medicine and electrical consumption, it has some of the widest range of engineering applications available to data analysts [3-5]. A plethora of new techniques have arisen in recent years to better facilitate the observance of previous trends and predicting the future. These techniques range of standard statistical methods such as moving averages (MAs) to genetic algorithms (GAs) and Neural Networks (NNs) [2].

Electricity consumption prediction is among the most important applications for time-series forecasting [6]. In providing insight into future load requirements, these methods help facilitate adequate power generation and distribution nation-wide. In the past, all manner of forecasting methods have been applied to electricity consumption, although a preference towards automated machine learning techniques have surfaced in recent years [6].

With the development of every new technology, the capabilities of data analysts to predict the future has increased. Since the popularisation of automatic machine learning (AutoML) pipelines, training and fitting data has become widely available to the general population [7]. Due to the generalisability of many time-series concepts, several AutoML models have been successfully deployed that can perform forecasts on any given input data [3].

However, the ease of use introduced by these pipelines brings with them a lack of customisation and understanding for the consumer. As such, these pipelines provide limited assistance in developing accurate forecasting models and can often become a ceiling for any user's capabilities. This problem gives rise to the need for an AutoML pipeline that is not only fully customisable but provides accurate feedback. Creating such a pipeline will help put control back in the user's hands, thus allowing for the creation of faster, more versatile, and better forecasting models.

2.2 Project Overview and Scope

This thesis is an exploration into the world of AutoML pipelines. Focussing on the electrical consumption characteristics of various datasets, this thesis will develop algorithms both from scratch and with the help of open-sourced Python libraries. The end goal of this thesis will be to develop an AutoML pipeline capable of time series regression on any input data. The key performance indicators for a successful pipeline are that it must be:

- Fast;
- Accurate;
- Able to work with any time-series dataset;
- Usable with any computer set-up;
- Customisable; and
- User-friendly.

In essence, anyone with any level of experience should be able to use the final pipeline on any time-series data for fast and accurate results. This will be approached using two separate algorithms, which are detailed below.

- Creating a hybrid grid search (GS) / Bayesian Optimisation (BO) pipeline from existing libraries. This pipeline will employ grid searches for optimal data cleaning and feature adding parameters, before feeding them into an open-sourced BO algorithm for fitting on neural networks.
- The BO pipeline will be contrasted against a self-made genetic algorithm (GA) pipeline. The GA will combine the GS and BO components of the previous model to provide accurate models with minimal train time.

The GA is intended to be a successor of the GS / BO algorithm, and as such will address the constraints and limitations imposed by it. The end goal will be to create a fully automated ML pipeline capable of achieving the aforementioned key performance indicators by automating the following steps:

- Data cleaning and feature adding
- Model creation and training
- Performance analysis and metrics acquisition
- Python file creation with the optimised pipeline
- Web dashboard visualisation and reporting
- Various user-controlled parameters for greater controllability

All constructed algorithms will be compared against a baseline persistence model. Tests will be performed on three electricity consumption and demand datasets: AEMO's NEM dataset [8], the GEFCOM 2012 Load dataset [2] and MathWork's Electricity Consumption Case Study dataset [1]. Each of these three datasets will be tested on forecasting horizons of 1-step, 1-day and 1-week, for a total of 9 trials.

2.3 Report Outline

The report is broken into 5 Chapters and an Appendices

- **Literature Review** provides a background into the prior art of time series forecasting. It will explore the previous efforts of data analysts within this field.
- **Theory** provides the necessary background knowledge required to gain useful insights from the process and findings from this thesis.
- **Methodology** details the step-by-step process used to create each of the AutoML pipelines.
- **Discussion and Results** will explore the performances of both pipelines against a variety of datasets and forecasting horizons.
- **Recommendations and Conclusion** will conclude with the findings of the report, as well as recommendations for future improvements and directions of research that could be of interest to the scientific community.
- **Appendices** shows some example outputs from both algorithms, as well as source code examples from their implementations.

3 Literature Review

3.1 Smart Meters

A smart meter is a device that is capable of logging and transmitting the energy consumption at the place of installation. At the end of June 2022, more than 29.5 million smart meters were being used in UK homes, with 45% of all domestic and retail meters being either of the smart or advanced variant [9]. While Victoria has boasted an uptake rate of close to 100%, other regions are lagging. This has indicated an inconsistency in the adoption rate of smart metering. Western Australia currently has an uptake rate of around 20%, and smart metering is not readily available within the Northern Territory[10]. As it stands, only 25% of mainland Australian homes are currently equipped with Smart Meters. The Australian Marketing Energy Commission (AEMO) have introduced systems to increase this to 50% of all Australian households [10].

The usefulness of smart meter data is twofold. Such data can be used by consumers to inform and optimise their consumption habits. It is equally as instructive to energy providers as insight into the trends of energy usage throughout a given day [11].

Smart meter datasets become ideal candidates for future predictions for several reasons. First and foremost, with a comprehensive energy reading being taken every 30 minutes, these datasets can provide accurate data with frequent intervals for any time of day[9]. However, although the data is high-frequency, high-quality, clean and used in 3.3 million nodes across Australia, very little of it is available to the public domain due to privacy and ethics concerns [12].

3.2 Feature Adding and Noise Reduction in Time Series Forecasting

Arora et al [13] discovered in their research into Conditional Kernel Density estimation on smart meter data that household consumption follows several distinct trends. Most notably, seasonal trend lines can be observed on an intraday, intraweek and yearly basis [13]. Figure 1 shows an example of the seasonal trends observed in a household's consumption over a 3-week period. These intraday and intraweek trends can be highly instructive for prediction on a short-term basis (24 hours). However, for longer term predictions they lose their efficacy and become counter-productive for model accuracy [1].

Household energy data tends to feature non-Gaussian distributions and widely varying rates of consumption [13]. Due to this, while it is possible to accurately predict the trend of power consumption, having an exact and accurate prediction is close to impossible. One common approach to mitigating this problem is to cluster predictions via KNN or Kernel Density estimations (KD) [14]. Other common approaches involve using hierarchical clustering methods, such as agglomerative clustering [15].

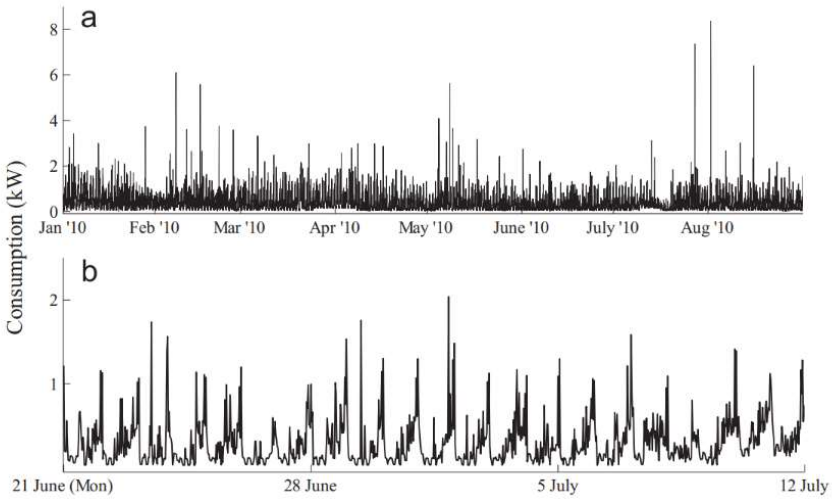


Figure 1: Consumption for a residential consumer for (a) 8 months and (b) a three-week period. From "Forecasting Electricity Smart Meter Data using Conditional Kernel Density Estimation" by S. Arora, 2014

Recent advances in weather prediction have also played a vital role in load forecasting predictions. Regnier [16] observed that weather indicators such as wetbulb and drybulb temperature, humidity and precipitation have shown to have direct correlations with energy requirements. Furthermore, the integration of holidays and weekends also prove to be highly instructive towards model accuracy, seeing as households will consume more power on the days when inhabitants are at home rather than work [1]. The inclusion of this data have become common methods of feature adding to improve predictive capabilities of forecasting models [16].

Although the inclusion of weather data, energy production, seasonal trendlines and holidays have served to increase the accuracy of model predictions, forecasting power consumption on a household basis is difficult to perfect. Apart from unforeseen outages, creating predictions on a suburb, city or grid-scale energy demand is relatively easy due to the stability of data [17]. Ma [17] observed that, when this focus is scaled down to a household basis, a high variability is introduced due to different lifestyles and day-to-day ongoings of residents.

Extensive research has been poured into reducing this uncertainty. In their studies into KD estimation of household consumption, Arora et al [13] applied a probability density function to the model outputs rather than a single prediction value. This acknowledged the variable nature of day-to-day life and energy requirements for household inhabitants, and logically concluded that it "Seems appropriate to provide a forecast of the probability density function for consumption recorded by a smart meter, rather than just producing a point forecast" [13].

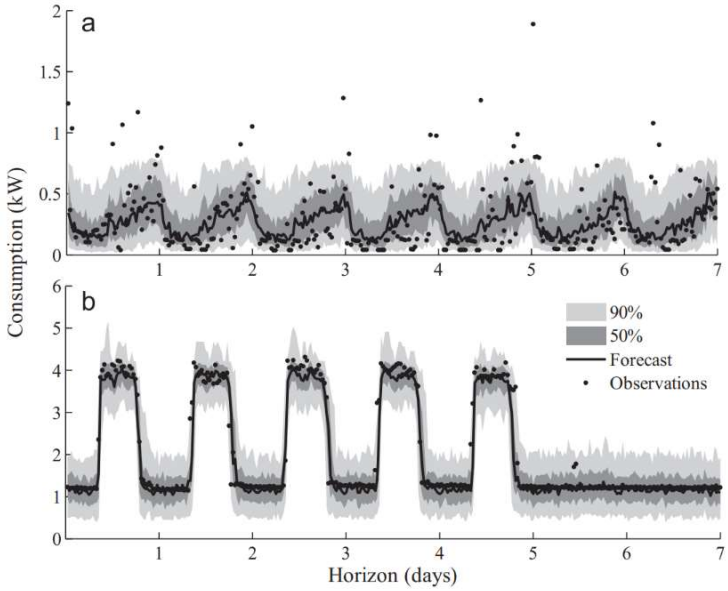


Figure 2: Summaries of density forecasts for (a) a residential consumer and (b) an SME. From "Forecasting Electricity Smart Meter Data using Conditional Kernel Density Estimation" by S. Arora, 2014

Figure 2 shows the final predictions of their Kernel Density model, which was able to provide a forecasted output along with 50% and 90% quantile ranges [13]. As can be noted, the model was able to accurately estimate all small to medium enterprises (SME), but unable to fully predict forecasts on a household scale. This serves to highlight the extreme variability of data in this problem.

In conclusion it was found that kernel-based methods outperformed simple baseline predictions models, and generated outputs that were comparable to exponential smoothing methods used in more sophisticated benchmarks [13].

3.3 Role of Genetic Algorithms in Optimising Forecasting Models

From economic modelling and regression to playing games, GAs are among the most widely-used and versatile hyperparameter tuning algorithms. In 2018, Loussaief and Abdelkrim employed GAs to optimise the Neural Architecture Search (NAS) of Convolutional Neural Networks (CNNs) [18]. This study used a GA to tune the number of convolutional layers and filters, as well as their sizes in each layer. Using a manually tuned CNN as a baseline, they were able to reach 30% accuracy on image recognition [18]. However, the automation process was able to increase the accuracy of image recognition predictions from between 90% and 98.94%, depending on the CNN architecture used [18]. Loussaeif and Abdelkrim concluded from these findings that GAs, and AutoML applications in general, outperform manual tuning both in terms of time taken and accuracy attained.

In another image recognition study, Liashchynski and Liashchynski [19] compared GAs to random searches (RS) and grid searches (GS) for the use in creating NAS for CNN models. These optimisation processes were employed on the CIFAR-10 dataset, and comparison results were based on execution time and accuracy of the proposed models. Their final findings showed that GAs were able to achieve an accuracy of 86% using 4.13 hours of runtime, as opposed to the 86% accuracy after 5 test runs from RS and 83% accuracy given 4.3 hours of runtime for GS [19]. Liashchynski and Liashchynski ultimately concluded that GAs were able to outperform both GS and RS consistently, thus making them the most effective model with respect to accuracy and runtime [19].

The findings published by Liashchynski and Liashchynski [19] indicate that GAs can perform faster and more accurately than many other hyperparameter tuning methods. These findings align themselves with the sentiment of Loussaeif and Abdelkrim's [18] work. Although both publications focussed on image classification problems, their findings should be transferrable to regression and time-series forecasting. Ultimately, both research papers support the decision to integrate GAs into time-series applications.

Apart from image recognition problems, the versatility of GAs have allowed them to be used in a range of novel applications. In 2002, Revello and McCartney [20] employed genetic algorithms to create strategies for war games. These simple naval blockade games pitted two players against each other: the red player being a benchmark model with a set strategy and a blue player using a genetic algorithm. Although all trials resulted in a GA that was able to achieve victory over the benchmark model, it was found that algorithms using elitism (including the parents from the previous generation) and counting (using GAs that had won their previous matches) produced the best results [20]. This delicate balancing of the GA's exploration and exploitation allowed it to achieve a top score of 95.97% [20]. This study has demonstrated the versatility of GAs to a wide array of problems. It also highlights the adaptive nature of GAs, and the importance of balancing its exploration and exploitation protocols to optimise performance. Although within a different field of research to electrical consumption, the observations from this paper will be critical in refining GA implementations.

In recent years, GAs have been successfully applied to a range of time-series problems. In 2020, Bas et al [5] applied ridge regressions and GAs to financial datasets. Using RMSE as the main evaluation metric, the researchers were able to compare Fuzzy algorithm implementations against Seasonal Auto-Regressive Integrated Moving Averages (SARIMA) and Winters Multiplicative Exponential Smoothing (WMES) baselines. The GA implementation far outperformed statistical approaches, achieving a final RMSE of 18.78 as opposed to 47.03 and 53.33 for SARIMA and WMES respectively [5]. With a final MAPE of 0.0348 [5], GAs were the best performing algorithms used within this research.

More in line with the aims of this thesis, Azadeh et al [6] were able to employ similar methods to predict electrical consumption within the Iranian Agriculture Sector. Within this article, the performance of an NN tweaked by a GA is compared against Auto-Regressive Moving Average (ARMA), Auto-Regressive (AR) and Moving-Average (MA) models [6]. The GA boosted NN model outperformed all other models in each trial. Table 1 compares the MAPE results from each of these models against the NN/GA implementation.

Table 1: Comparison of Statistical Models (Time Series) to the NN/GA implementation (ANN) proposed by Azadeh et al. (Sourced from [6])

Models	X_1	X_2	X_3
ANN	0.132469	0.1023	0.010582
Time series	3.157024	0.3822347	2.1188473

The respective works published by Bas et al [5] and Azadeh et al [6] show direct correlations to the aims of this thesis. Both works have highlighted the value of GA implementations on time-series data. Furthermore, a vast improvement in predictive capabilities has been observed compared to common statistical methods such as ARIMA. Finally, Azadeh et al [6] focussed their observations on electrical consumption within the Iranian Agriculture Sector. Successful implementations under both circumstances correlate well with applications on NEM and GEFCOM electrical consumption datasets.

3.4 Role of Bayesian Optimisation in Time-Series Forecasting

In 2023, Yan et al [21] were able to employ Bayesian Optimisation strategies to enhance the effectiveness of ML algorithms in diagnosing the long-term water quality of the Guangdong province. Their research delved into the use of five ML models, each employing Bayesian Optimisation for hyperparameter tuning. These models were then tested against key water pollution indicators including total phosphorous and dissolved oxygen to determine the long-term trend of water quality [21]. Through their work, it was found that an optimised stacked generalisation model performed best, with an accuracy of 0.992 and Kappa co-efficient of 0.987 [21]. Although these findings were based off classification problems, they still serve to highlight the effectiveness of BO algorithms in enhancing model accuracy.

Another time series application of BO algorithms was conducted by Colopy et al [22] to identify the deterioration of hospital patients based on measurements of the patient's vital signs. Applied against the data of thirty-four patients from the University of Pittsburgh Medical Centre, the BO algorithm was shown to enhance model performance in thirty-two of the thirty-four trials [22]. Graph (b) within Figure 3 compares the predictive capabilities of uni-formative and BO-tuned models, where every dot above the 45-degree line indicates an improvement over the uni-formative model. It succeeded for all patient datasets except the two that featured less than two hours of heart rate data [22]. Graphs (a), (c) and (d) compare the empirical cumulative distribution function performances of the models for future windows of 10-15, 45-60 and 1-5 minutes respectively [22]. Significant improvements were observed for long-term forecasts of greater than 10 minutes, although improvements were minor over a shorter future window.

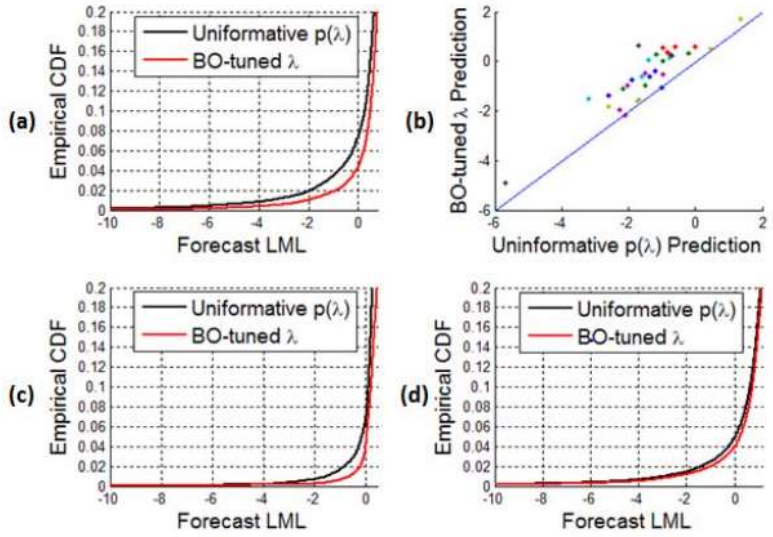


Figure 3: Empirical CDF comparisons of uni-formative and BO-tuned models for predictions of patient's vital signs. (a) provides a 10-15 minute prediction window, (c) 45-60 minutes and (d) 1-5 minutes (Sourced from [22])

The works published by Colopy et al [22] provide several important insights into the applications of BO algorithms in time series data. Firstly, their work indicated the BO methods are effective in improving model prediction. In line with the sentiments of Waring et al [23], they observed that AutoML methods are far more effective and time-efficient than manual tuning. An equally interesting observation is the effectiveness of BO optimisation based on the future window. The works of Colopy et al [22] indicate that BO algorithms outperform manual implementations by a greater margin as the future window increases. This would suggest that the improvements of BO algorithms will be more noticeable for longer term forecasts on electrical consumption data.

More in line with electrical consumption, Akil et al [24] employed Bayesian Optimisation strategies to load forecasting for household data. Using a variety of hybrid LSTM models, their work targeted the instantaneous power ratings of the household's heat pump (HP), electric vehicles (EV) and photovoltaic (PV) systems. The BO algorithm in conjunction with LSTM was found to be highly effective, producing RMSE values of 0.0893, 0.052 and 0.2629 for the HP, EV and PV systems respectively [24]. Figure 4 compares the best and worst predictions against the instant power values.

Although work of Akil et al [24] indicates that BO algorithms can assist in the prediction of electrical consumption, their work is not focussed on the optimisation capabilities of BO. Instead, their observations compare the performances of different LSTM architectures. Since all models featured BO optimisation, it is difficult to quantify its effectiveness. Throughout this thesis, emphasis should be placed on quantifying model improvement due to BO implementation.

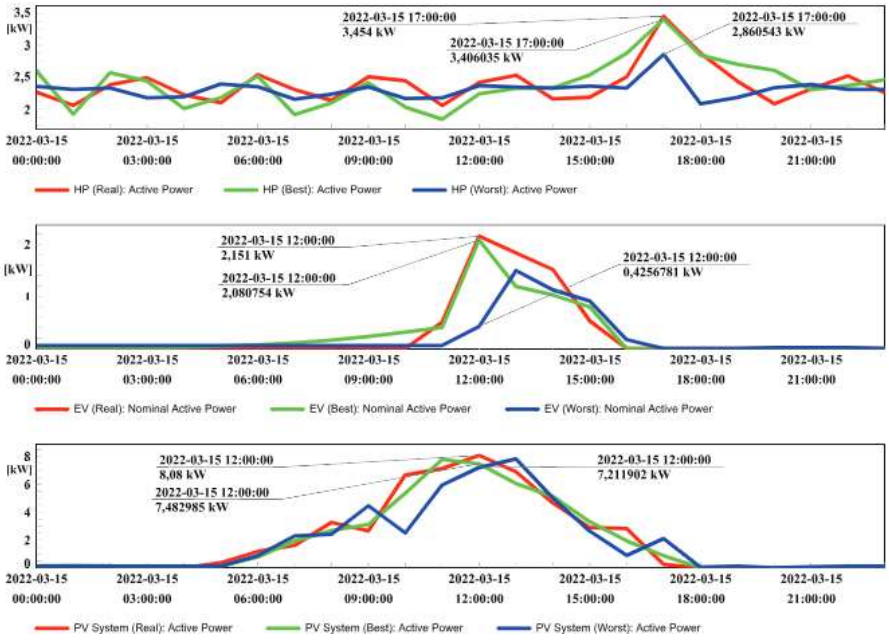


Figure 4: Instant power predictions of heatpump, EV and PV systems (Sourced from [24])

3.5 Commonly used Forecasting Models within GEFCOM 2012 and 2014

During IEEE's Global Energy Forecasting competition in 2012, 969 forecasting models were developed by 152 contestants to find the most accurate forecasting model [2]. Common approaches included multiple linear regressions (MLR), the Box-Jenkins approach and NN models. Of these implementations, models implementing gradient boosting machines and random forest models proved to be the most accurate on a 24-hour basis, having a root-mean-squared-error (RMSE) of 59'273 and 50'976 respectively; half that of many none ML-based implementations [2].

This sentiment was repeated in the 2014 GEFCOM competition, where teams that used ML architectures were able to generate results with relatively low error outputs. Zhang et al [14] were able to use K-nearest-neighbours (KNN) and KD clustering, much in line with the works of Arora et al [13] in 2014. This model was able to accurately predict load forecasts with a RMSE average of 0.15513 [14].

The findings of this meta-analysis serve to strengthen the consensus that ML models are capable of outperforming more conventional statistical methods [23, 25]. Combining this with the observations of Liashchynski and Liashchynski [19] as well as Loussaief and Abdelkrim [18] mirrors the thesis' sentiments of using AutoML pipelines to provide the accurate forecasts with minimal time investment.

4 Theory

4.1 Data usage and Feature Adding in Electrical Consumption Forecasting

As detailed within the literature review, several strategies exist in which time series data can be enhanced. Although this increases the size and complexity of training data, it also reduces overfitting and helps make ML models more robust to changes in data [1]. common examples of feature adding that will be explored within this thesis are:

- Weather data: Regnier [16] discussed the importance of including weather data to enhance the performance of time series forecasting. Common weather data includes the temperature and humidity, and are vital within the predictions of electrical generation and consumption, particularly for renewable energy resources [16].
- Holidays and working days: The day of the week will govern the activities undertaken by people. This in turn affects the usage of electricity within households and cities [1, 17]. Including details about the day of the week and whether it is a public holiday have been shown to increase the accuracy in future forecasting for time series models [1].

Secondly, there are many common methods of data augmentation, which will take existing data and alter it to extract more information from the given input. Commonly used methods of data augmentation of time series data include the use of moving average filters, seasonal decomposition and windowing.

4.1.1 Moving Average Filters (MAF)

One of the most common methods of noise reduction, moving average filters (MAFs) smoothen out changes in data by averaging them with previous values [4]. This method can be further improved upon by applying weights to the MAF which gives a higher precedence to more recent data. This method, known as exponential smoothing, was applied by Siddharth et al in their studies into kernel density clustering [13]. Variations of MAFs were commonplace in the earlier statistical methods of time series analysis, and were often used as they keystone components in models such as ARIMA, ARMA and SARIMA [4]. Although these methods have in recent years been eclipsed by ML [3], MAFs still serve as an important component within feature adding pipelines [5].

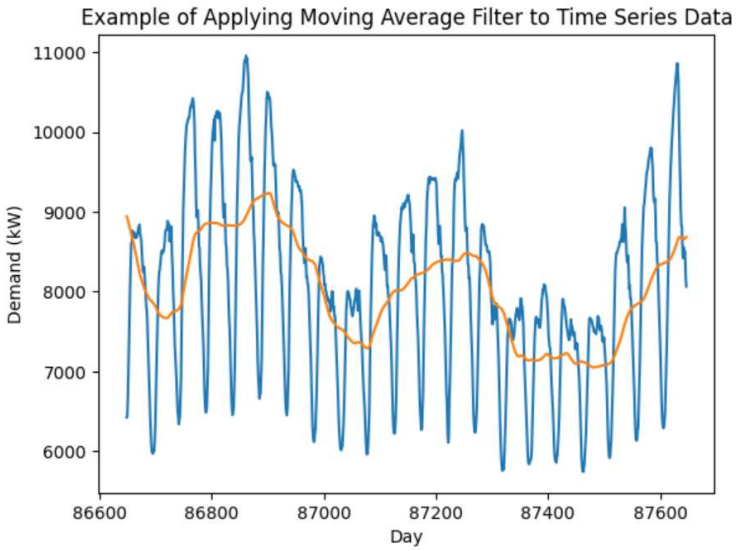


Figure 5: Example of applying moving average filter to Mathwork's load forecasting dataset (Deoras, 2016)

4.1.2 Seasonal Decomposition

Another commonly employed feature adding method, seasonal decomposition serves to break signals into their trend, seasonal and residual (noise) components. In the case where the data is known to have seasonal trends, this method is an effective method for isolating noise from a dataset [26]. As observed by Arora et al [13], energy consumption data has been observed to have intraday and intraweek cycles, thus making it an ideal candidate for seasonal decomposition. Figure 6 shows an example output from Statsmodel's seasonal decomposition model that is available within their python library [27].

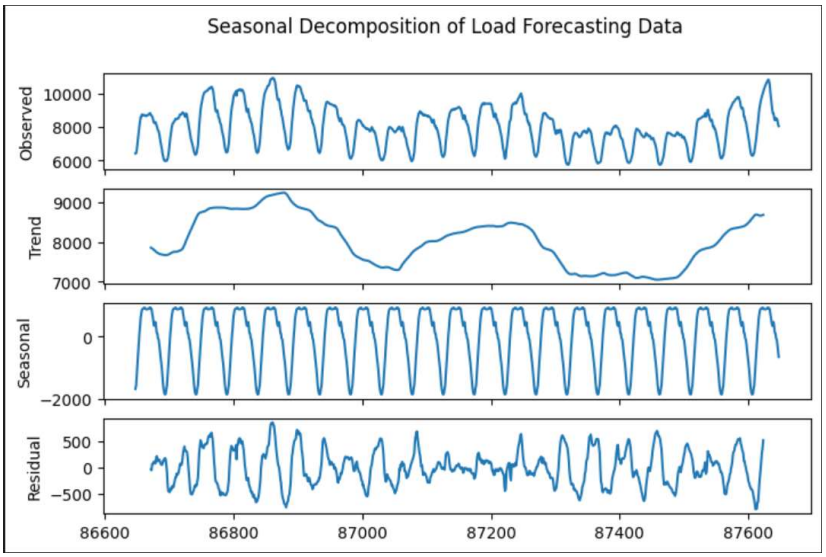


Figure 6: Seasonal Decomposition of Mathwork's Household Load dataset (Deoras, 2016)

4.1.3 Windowing

Windowing is a critical technique within time series analysis that involves segmenting data into smaller, sequential segments. These aptly names “windows” facilitate the extraction of meaningful patterns and capture temporal dependencies from within data [1, 6]. Sequential observations in time series data are often inter-related, and previous points in time can affect present or even future values. Windowing helps to better capture these relationships, thus ensuring that both short-term and long-term trends within time series data can be identified [1]. Figure 7 offers a visual representation of the application a 5-minute window to time series data.

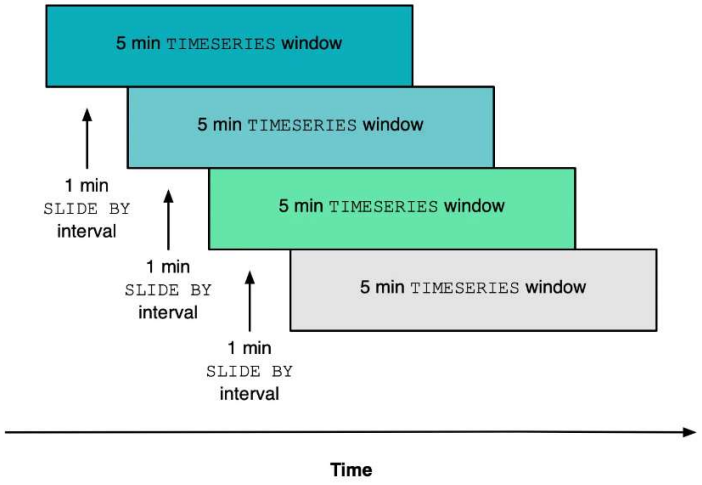


Figure 7: Representation of applying a sliding window of 5 entries to a time series dataset (Sourced from [28])

4.1.4 Principal Component Analysis (PCA)

Principle Component Analysis is a widely used dimensionality reduction technique. The primary goal of PCA is to transform the dataset into a new co-ordinate system using a set of orthogonal axes known as “Principal Components” [29, 30]. This is achieved through identifying the correlations between inputs. The identification of these principal components assists in ML applications in the following ways:

- Dimensionality Reduction: In isolating the components with the highest correlation, it becomes possible to remove extraneous inputs. This reduces training time, computational complexity, and memory consumption [29].
- Noise removal: In keeping only highly correlated data, noisy inputs are removed in the process [30].
- Pattern identification: The removal of noisy data better facilitates an ML model’s ability to extract patterns and relationships between the remaining principal components [29].

PCA functions by redefining the input data with respect to a set of orthogonal axes, known as "principal components." In doing so, PCA can emphasize important input components while reducing the size and complexity of the dataset [30]. Figure 8 shows a pictographic representation of the dimensionality reduction process of breaking a dataset down into its principal components.

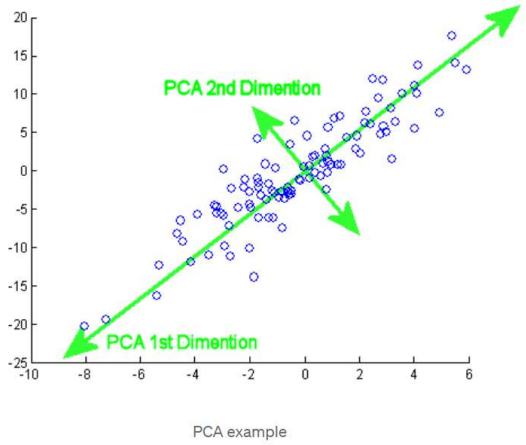


Figure 8: Example of PCA transforming the input data onto a new set of orthogonal axes known as principal components (Sourced from [30])

PCA has become the most popular method in dimensionality reduction and is regarded as among the most common methods of data cleaning [30]. Its applications are useful to a range of machine learning applications, time series forecasting being one of those.

4.1.5 Agglomerative Clustering

A popular method of hierarchical clustering, agglomerative clustering is a method of allocating similar datapoints into clusters. This hierarchical method of clustering allocates all datapoints within a set into clusters, ranked by the strength of their similarities [15, 31]. Clustering is typically applied in time series forecasting to uncover complex relationships that appear in temporal data. By clustering these datapoints by their similarities, it helps to distinguish between distinct data patterns, helping ML models identify critical relationships within data [15].

Figure 9 shows an example of hierarchical clustering being applied on a set of data.

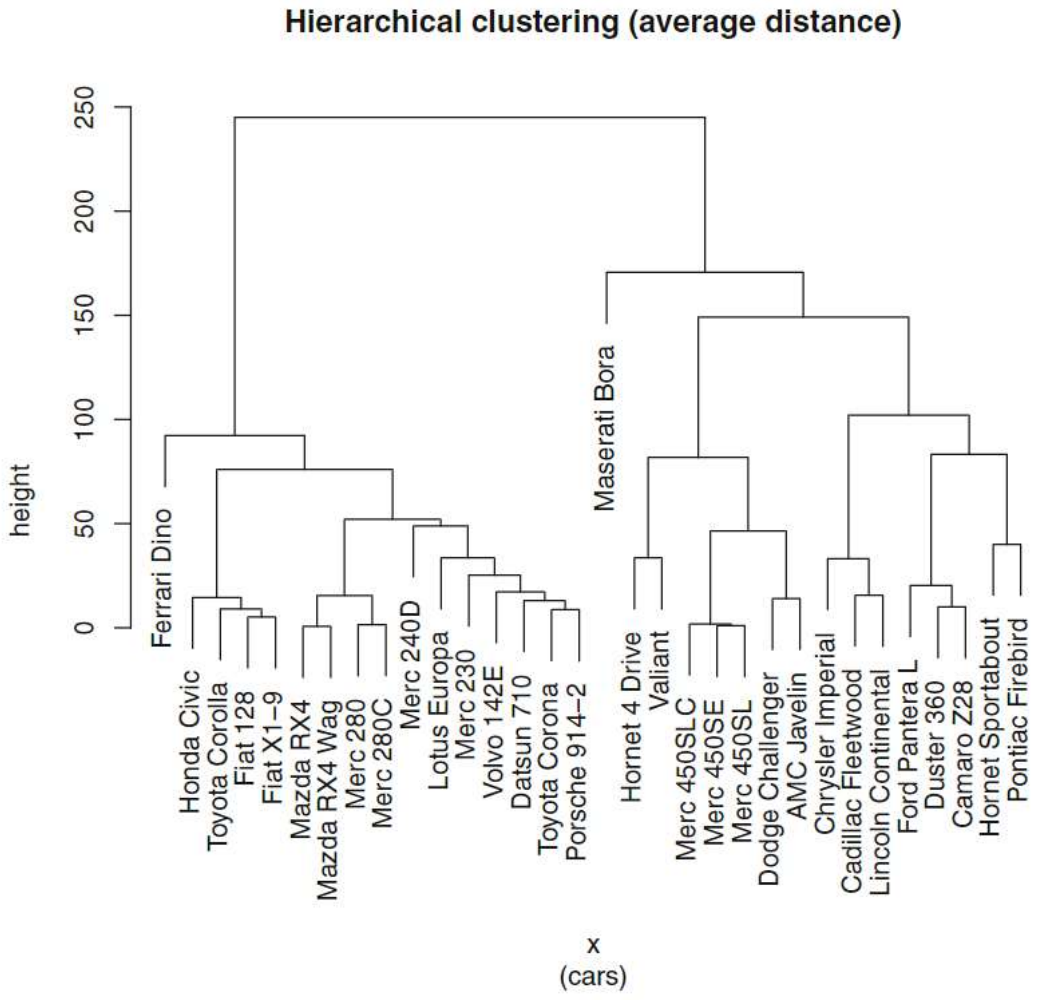


Figure 9: Example dendrogram created by clustering cars (Sourced from [15])

4.2 Principles of Neural Networks

Neural networks (NN) are a subordinate field of machine learning that are inspired by the design and architecture of biological brains [32]. NNs contain a set of internally interlacing neurons that are designed to compute values from inputs, in a method akin to the functioning of biological brains [3]. The non-linear nature of NNs make it ideal for extracting complex features and relationships between inputs and outputs. Figure 10 shows the architecture of a basic neural network.

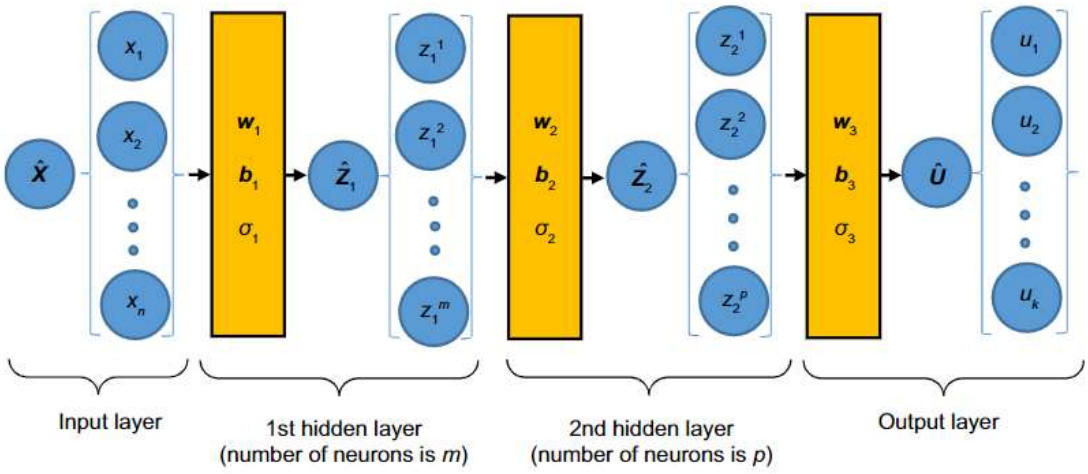


Figure 10: Architecture of Neural Network (Sourced from [32])

All NNs contain a set of parameters and hyperparameters that control the process of training and predicting on data. The tweaking of these parameters alter the behaviour and performance of the ML model. Some of the hyperparameters that can be adjusted within all neural networks include:

- The number of hidden layers: reducing the number of hidden layers correlates with a fast and well-generalised NN, at the expense of losing accuracy [3]. In increasing the number of hidden layers, the NN will become better able to classify inputs, although a point of saturation is reached where the model becomes more complex without increasing its accuracy [3]. For this reason, it can often be inconvenient to design a densely layered NN.
- The number of neurons present on each hidden layer: Similar to increasing the depth of the NN, increasing the width of each layer will add more accuracy by giving the NN a better ability to extract information from input data [32].
- The activation functions: Changing the activation function will alter the output of the neuron regarding its input and connection's weights [3].
- The learning rate: Having a higher learning rate will cause the NN to learn faster, although this introduces the chance of overshooting an optimum solution. Adjusting the learning rate hyperparameter will help create the fastest learning environment to reach the most optimised solution [3].
- The number of epochs: Increasing the number of epochs will train the model for a longer time frame. In doing this, the accuracy will be increased, although this will eventually

become meaningless as the increased training time will result in smaller returns in accuracy [3].

- The batch size: Determines the number of samples to be propagated through the neural network. Increasing the batch sizes, similar to epochs, will increase accuracy at the expense of longer runtime [3].
- The presence and rate of dropout layers: To prevent the system from overfitting, dropout layers randomly remove nodes from the training and backpropagation process [17]. Properly tweaking dropout rates eliminates overfitting while minimising accuracy reduction due to the removal neurons.

4.2.1 Long Short-Term Memory Networks (LSTM)

LSTM are a type of hidden layer derived from recurrent neural networks (RNNs) that are designed to deal with long-term dependencies [3]. Using its ability to save information in a 'memory cells,' it can recall this memory and use it with present information as needed. In predicting series data, LSTM can outperform NNs and RNNs due to this long-term storing of memory in cells. Figure 11 shows the architecture of an LSTM layer.

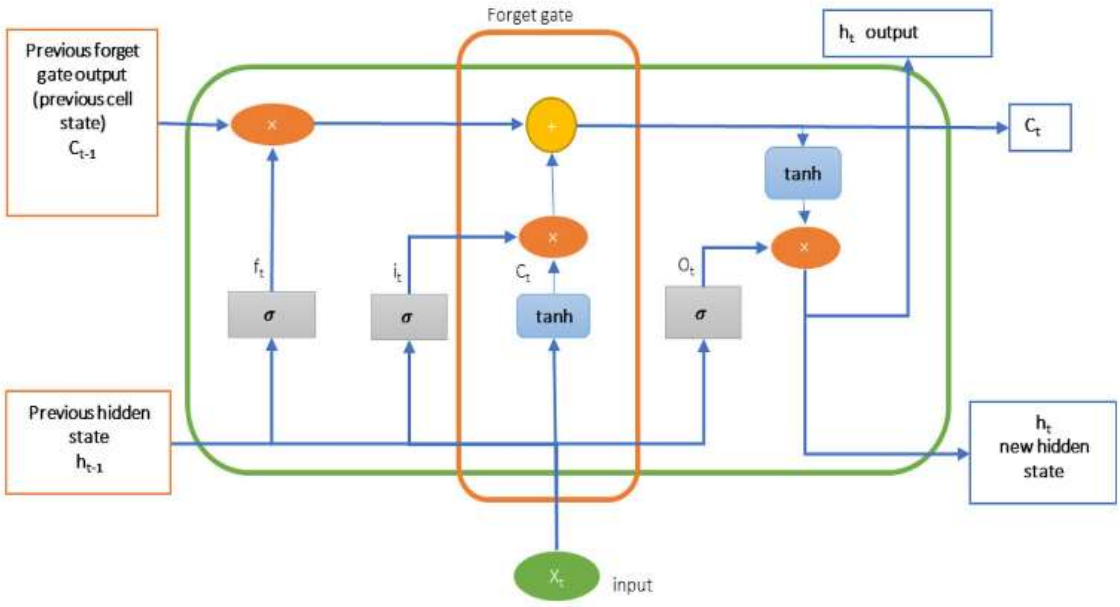


Figure 11: Architecture of LSTM Memory Cells (Sourced from [3])

4.2.2 Convolutional Neural Network (CNN)

Inspired by the functioning of visual perception, CNNs are a kind of feedforward network that employ convolution to extract features from data [33]. The following key features separate the architecture of a CNN to conventional NNs:

- Each neuron is only connected to a local branch of neurons, rather than the entire previous layer. This is effective in reducing parameters and speeding up convergence [33].
- Groups of connections can share the same weights, which further works to improve parameter reduction [33].

The architecture of a CNN can be found within Figure 12.

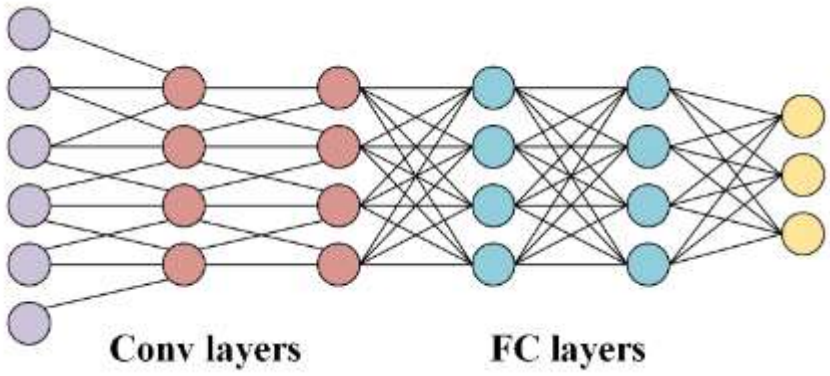


Figure 12: Architecture of CNNs (Sourced from [33])

4.3 Optimisation Strategies

Automated Machine Learning (AutoML) is a method of solving ML tasks such that little to no human intervention is required. This provides non-experts with the resources to apply machine learning techniques to specialised tasks with minimal prior knowledge [3, 25]. AutoML commonly seeks to fully automate the process of optimizing:

- Model selection: Given a selection of models and a dataset to train on, AutoML seeks to find the best-fit model for the data. Operating without human intervention, AutoML is able to iterate through a selection of different models to train on the same data input, before returning the best performing model [25].
- Hyperparameter optimisation: As detailed within the theory section for neural networks, hyperparameters completely govern the behaviour of a machine learning model. As such, by automatically tweaking the values of hyperparameters, AutoML models have the ability to optimise a given ML model to best suit the input data [3].

There are two common forms of Automated Machine Learning that will be explored within this thesis: Bayesian Optimisation and Genetic Algorithms.

4.3.1 Bayesian Optimisation

Bayesian Optimisation is a powerful and efficient method of optimizing complex systems. Due to its versatility and ability to evaluate blackbox functions, it is an ideal candidate for use in ML and hyperparameter tuning [24]. Some of the benefits of BO are:

- Efficiency – BO is a lightweight solution that is popularised due to its fast evaluation times [24].
- Robustness – Due to its inherent functionality of optimising model performance, BO algorithms are highly resistant to noise and can interpret hidden relationships within datasets [24, 34].
- Adaptability – BO algorithms can evaluate any black box function. As such, this versatile algorithm is employable over a wide range of applications. Hyperparameter tuning is simply one of the most common applications [34].
- Fewer function evaluations – Due to its intelligent optimisation process, BO is able to hone in on the most efficient solution in fewer function calls, as opposed to GS or RS [22, 34].
- No prior knowledge – In attempting to fit a black box function, BO strategies require no prior knowledge of the data they are working on [34].

In essence, Bayesian Optimisation attempts to design its own function to recreate the black box function it is working around [34]. Figure 13 shows that this is achieved by drawing points from the black box function and mapping these values to its own surrogate function.

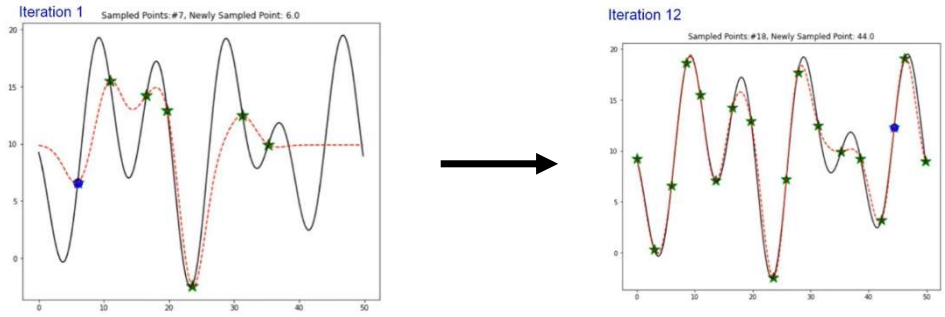


Figure 13: Description of BO's process of fitting black box functions (sourced from [34])

The process behind employing BO is an iterative sequence of mapping values to its surrogate function. The steps undertaken within Bayesian Optimisation are:

1) BO begins with a black box function, known as the objective function. In the case of this thesis, the black box represents the electricity consumption data the BO is attempting to fit to.
2) An initial surrogate model is created. The surrogate model is the BO algorithm's attempt at recreating the black box function. These models range in complexity from simple Gaussian Process models up to Bayesian Neural Networks [22, 35].
3) An acquisition function is used to determine where the next point of exploration should be. The acquisition function balances exploration and exploitation of pre-existing points within the surrogate model to determine the most effective search spaces for model optimisation [34]. Upper Confidence Bound (UCB) is among the most commonly used acquisition functions [35].
4) The point selected by the acquisition function is then evaluated. The surrogate model is then updated appropriately to facilitate the newly found evaluation point [34].
5) Steps 3 and 4 are repeated indefinitely, refining the surrogate model with every new iteration.
6) The final model created through the Bayesian Optimisation process represents the global optimisation within the search space provided to the algorithm.

Figure 14 shows a flowchart version of the BO algorithm's process.

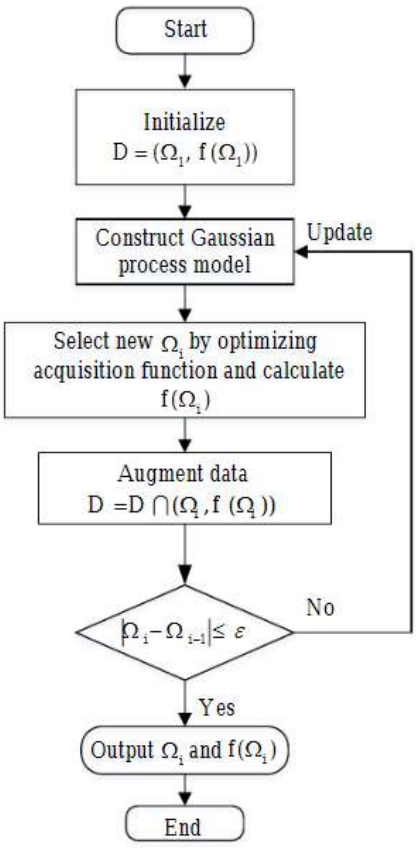


Figure 14: Flowchart of Bayesian Optimisation process (Sourced from [35])

4.3.2 Genetic Algorithms

Genetic Algorithms replicate natural evolutionary processes to provide optimised solutions within a given search space. This function occurs through the creation of a population, filled with randomised genomes. The benefits of employing a GA are as follows:

- Global search – Similar to GS and RS, GAs can effectively sift through an entire search space. However, the utilisation of the evolutionary process allows for an optimised approach, reducing the number of searches that must be used to reach an optimum output [20].
- Parallelism – Unlike many other hyperparameter tuning methods, GAs are inherently parallel processes. This allows them to be run in multiple threads or processes [36].
- Adaptability – Evolutionary processes allow the GA to be applied to a wide range of problems, including regression, economic modelling and even game-playing [37].
- Robustness – Due to the extensive evolutionary process, GAs are resistant to noisy inputs and can interpret hidden relations within datasets [18-20, 32].
- No prior knowledge – GAs require no prior knowledge of the data being input. This is what attributes to their versatility and adaptability in a range of applications [36].

Figure 15 provides a flowchart detailing the evolutionary process employed by GAs.

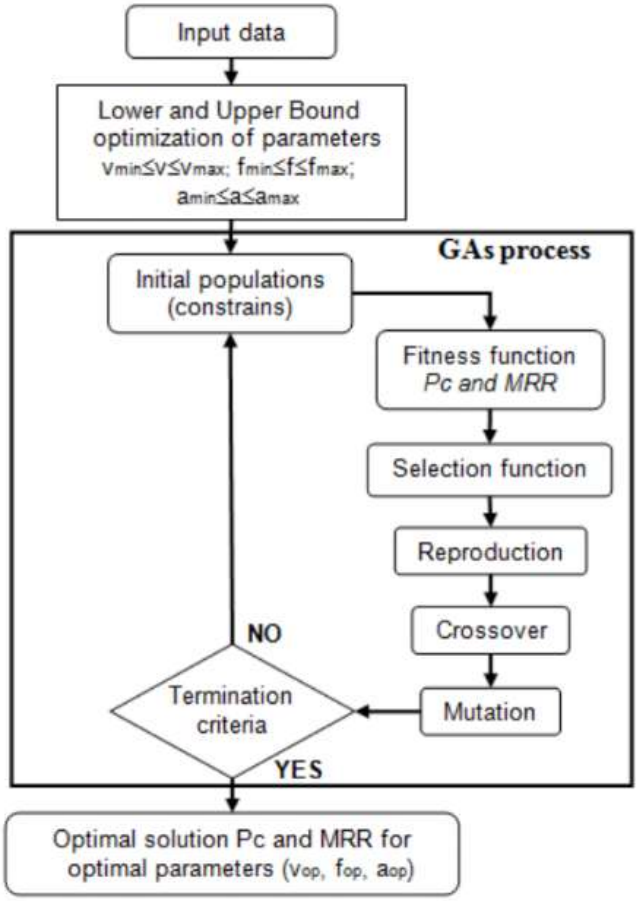


Figure 15: Flowchart showing the evolutionary process employed by GAs (Sourced from [37])

Although differences exist within every GA implementation, three main characteristics are universally present within every approach [20]. Fitness calculation, the first of these characteristics, refers to the utilisation of a fitness metric with which to calculate the success of each genome created [36]. Diversification is the second trait and refers to the model's exploration of an entire solution domain and targeting of solutions with the greatest optimisation potential [20]. The final characteristic, exploitation, refers to the GA's ability to intensify searches on existing solutions, and attempting to optimise on these hyperparameters. Alternating between exploration and exploitation is what attributes to the success and efficiency of GAs compared to RS or GS [19].

As shown in Figure 15, fitness functions are the first step within the iterative procedure of creating successive generations. Fitness functions provide the metric with which a GA can determine the success of each genome created [18-20]. Within natural evolution, the fitness function is determined by an organism's ability to survive and reproduce. Within GAs, this function can take myriad forms. Similar to the works of Azadeh et al [6], this thesis measures a genome's ability to forecast electrical consumption as its fitness. Following the calculation of each genome's fitness, exploration and exploitation are employed to create a new generation of genomes based off the most successful models.

Exploration and exploitation are handled within the GA's selection, crossover, and mutation functions [20]. After the fitness of each genome has been measured, the selection function handpicks the fittest genomes to serve as the parents for the next generation. Although it is universally accepted that the highest scoring genomes should be used most prevalently as the parents [20], several GA implementations allow the re-introduction of lower scoring genomes into successive generations [32]. Doing so increases the exploration of potential solution spaces, at the expense of reducing the exploitation of optimised genomes [18, 32]. The effectiveness of a GA hinges on the balancing of exploration of a diverse array of genomes with the exploitation of successful genomes within the selection function.

Following the selection of parents to form the next generation, cross-over techniques are employed to create a range of child genomes. This method of exploitation involves merging the genetics from parent genomes to create unique offspring [37]. Figure 22 provides a pictographic representation of this process. GAs utilising more parents can create more unique variations of offspring, therefore having a greater propensity for exploitation of successful genomes [37]. This process of exploration is generally balanced out using the GA's final evolutionary procedure: the mutation function.

The mutation function combines exploration and exploitation by slightly mutating the offspring generated by the cross-over function [32]. By randomly tweaking some genetics within each genome, the GA is able to explore a wide range of effective solutions [20]. Similar to increasing the learning rate of an NN, mutating more genetics at one time allows for faster evolution, at the expense of making it difficult to reach a global maximum optimisation [18, 32]. Manual adjustment of mutation rates is generally required to balance training time with model optimisation [18]. Lastly, rates of mutation within each successive generation are balanced with cross-over rates. In doing so, exploration and exploitation of genomes can be tuned to reach the optimum efficiency [32].

5 Methodology

5.1.1 Dataset and Preprocessing

There were several significant advantages to the GEFCOM 2012, Mathworks and AEMO datasets that made them easy to work with. The main of these advantages were:

- Public availability: In being released freely to the public, all datasets could be used without risk of incurring privacy ethic violations.
- Cleanliness: Although some degree of data cleaning was necessary for each dataset, they were made highly useable before being released to the public.
- Correlation: All datasets, being concerned with electrical demand and consumption, had a high correlation in the data inputs being used. As such, there was a large crossover in the process of developing an ML pipeline for the datasets.

These datasets were not identical. Differences in the structures of the datasets made it necessary to employ uniquely tailored preparation processes before they were ready for feeding into the respective AutoML models. Although the Mathworks dataset could be deployed as-is, the GEFCOM and AEMO datasets had to undergo the following pre-processing steps.

5.1.1.1 GEFCOM 2012 Data

The GEFCOM 2012 data was relatively sparse in terms of inputs. Data was provided for the temperature and hourly load consumption (in kW) for a US utility with 20 zones. The dates for this data ranged from the 1st hour of 1/1/2004 to the 6th hour of 30/6/2008. Participants of the original competition were required to forecast at both the zonal level (for each of the 20 zones) and the entire system (The sum of the 20 zones). The performance of a benchmark model was also provided to compare against. Figure 16 provides an example of the original data for hourly loads. Note that the structure for temperature data was identical.

	zone_id	year	month	day	h1	h2	h3	h4	h5	h6	...	h15	h16	h17	h18	h19	h20	h21
0	1	2004	1	1	16,853	16,450	16,517	16,873	17,064	17,727	...	13,518	13,138	14,130	16,809	18,150	18,235	17,925
1	1	2004	1	2	14,155	14,038	14,019	14,489	14,920	16,072	...	16,127	15,448	15,839	17,727	18,895	18,650	18,443
2	1	2004	1	3	14,439	14,272	14,109	14,081	14,775	15,491	...	13,507	13,414	13,826	15,825	16,996	16,394	15,406
3	1	2004	1	4	11,273	10,415	9,943	9,859	9,881	10,248	...	14,207	13,614	14,162	16,237	17,430	17,218	16,633
4	1	2004	1	5	10,750	10,321	10,107	10,065	10,419	12,101	...	13,845	14,350	15,501	17,307	18,786	19,089	19,192

Figure 16: First 5 inputs of the hourly load consumption data

Both the BO and GA algorithms required data to be input in a specific format. To prepare the GEFCOM data, it was necessary to transpose the dataframe and have each of the rows transformed into columns. Secondly, it was necessary to have a datetime row which contained the date of each input. None of the information was changed during pre-processing: the sole process was matrix transposition. Figure 17mshows the GEFCOM load data post-processing and ready for feeding into the ML pipelines.

Date	Hour	Load
2004-01-01	0.0	16853.0
2004-01-01	1.0	16450.0
2004-01-01	2.0	16517.0
2004-01-01	3.0	16873.0
2004-01-01	4.0	17064.0

Figure 17: GEFCOM load dataset after matrix transposition

5.1.1.2 AEMO Data

Data gathered from AEMO's NEM web contains a range of information on the current condition of Australia's energy grid. For this thesis, NEM data from January until May in 2022 was used. Upon inspection it was found that a great deal of sparsity existed within the dataset of dispatch and pre-dispatch forecast tables, allowing for most columns to be removed inconsequentially. Furthermore, it was found that many of the remaining columns were for indicators not relating to the state's electrical demand. As such, a cross-correlation [38] was conducted and only the most relevant columns of information were retained. A list of the retained columns is as follows [8]:

- RAISE6SECACTUALAVAILABILITY: The 6 second availability for upward Frequency Control Ancillary Services (FCAS).
- RAISE5MINACTUALAVAILABILITY: The 5-minute availability for upward regulation services.
- LOWER6SECACTUALAVAILABILITY: The 6 second availability for downward regulation services.
- UIGF: Unconstrained Intermittent Generation Forecast.
- SEMISCHEDULE_CLEAREDMW: The cleared semi-schedule generation capacity in MW.
- SS_SOLAR_UIGF: Solar UIGF.
- SS_SOLAR_CLEAREDMW: Cleared semi-schedule solar generation in MW.
- TOTALDEMAND: The target demand.
- Dates: Date and time of data input.

Initially, tests were conducted on the NEM dataset that only included the TOTALDEMAND, which is the target electrical demand, and the date and time at which the demand was recorded. In later stages, the dataset was increased to accommodate all the above inputs, which increased accuracy at the cost of longer runtime.

5.1.2 Data Cleaning and Feature Engineering

Following the preparation of all three datasets, they were ready for feeding into both pipelines. It is worth noting that both the BO and GA algorithms implemented similar data cleaning and feature adding processes. As such, this section will address the first stage of both pipelines, which included the augmenting of said data.

Before feature adding could occur, there were several data cleaning techniques that had to be employed. Although mainly concerned with the removal of empty data points, the steps taken within the data cleaning process can be found in Table 2. Some methods of data cleaning appear only in the GA implementation due to them being introduced after the completion of the BO algorithm.

Table 2: Adjustable inputs over the data cleaning process.

Step	Description	Values
Imputing	Selects the method of imputing to be applied to null values	[Mean, Median, Mode, None]
Outlier Removal (GA only)	Allows removal of values outside of 2.5 standard deviations	[True, False]
Scaling	Selects the scaling method to be applied to datasets	[Standard, Robust, Normalizer, None]
Dimensionality Reduction (PCA)	Creates a PCA. The number of components can be decided using Minka's MLE [39]. SVD Solver can be randomised in line with the works of Halko et al [39].	[Minka's MLE, Randomised, None]
Agglomerative Clustering (GA only)	Allows clustering of data points	Linkage: [Ward, Complete, Average] Metric: [Euclidean, L1, L2, Manhattan, Cosine]

Although both ML pipelines included similar data cleaning functionalities, they differed in terms of implementation. The BO algorithm, being the first algorithm created, implemented a grid search over all available parameters before selecting the most successful data cleaning pipeline. This ensured an optimised pipeline at the cost of extremely long run-times. In fact, the extended runtime served as the main motivator for creating the GA to supersede the BO implementation. In contrast, data cleaning was implemented within the genome of the genetic algorithm, thus ensuring that an optimised pipeline would be achieved with far less overhead than the GS implementation.

After the data cleaning process, the following features were added:

- PrevEntry: The lagged target value from the previous entry (1 step behind)
- PrevDaySameHour: The lagged target value from a day before
- PrevWeekSameHour: The lagged target value from a week before
- Prev24HourAveLoad: MAF over the past 24 hours
- Weekday: The day of the week
- Holiday: A flag as to whether it was a weekend or a public holiday (i.e. Non-working day)

Furthermore, a seasonal decomposition was implemented to include the following features:

- IntraDayTrend: The daily trend. This represents the long-term movement of the data.
- IntraDaySeasonal: The daily seasonal cycle.
- IntraWeekTrend: The weekly long-term trend of the data
- IntraWeekSeasonal: The weekly seasonal cycle

The residuals for the daily and weekly seasonal decomposition were not included, as they represent noisy components of the data's trend.

5.1.3 Performance Metrics

The final similarity shared between the BO and GA implementation was the recording of metrics. Both algorithms employed the following methods of recording their performance:

- K-fold Cross-Validation: The potential for inaccuracy when working with ML models introduces the necessity of using cross-validation [38]. As such, K-fold cross validation was included in both the BO and GA algorithm. The GA algorithm allows for manual adjustment of the number of cross-validations. In contrast, the BO algorithm was fixed at 5-fold CV.
- R2: Used as a final indicator for accuracy, R2 was included within both pipelines. The GA further expanded upon this by using R2 as the cost function. It was selected over other cost function options because model performance is directly proportional with an R2 value, with a value of 1 indicating a perfect forecasting. This was simply more conducive with the GA architecture than other options such as RMSE.
- MAE: Including the mean-absolute-error as a performance metric would provide an unbiased reflection of model performance that focussed on absolute differences. It was important to include MAE in tandem with RMSE, as MAE is more robust against the effects of outliers and will not be as heavily influenced by large error margins.
- MAPE: Similar to the mean-absolute-error, including the MAPE provided a percentage of error, which alongside R2 is the easiest metric to gauge the effectiveness of a model.
- RMSE: Including the mean-squared-error gave valuable feedback on the presence of outliers and large errors within a model's predictions. This could be used alongside the MAE to gauge whether the magnitude and frequency of the errors made by the model.
- Time: The runtime for each algorithm was also recorded. The purpose of this thesis was to create a pipeline that could quickly and effectively create an accurate model. Runtime was a critical metric that served as one of the core components in each pipeline's construction.

MSE was ultimately not included as a metric because it provided little more information than was already given by MAE and RMSE.

5.1.4 Bayesian Optimization Pipeline

By incorporating the above data cleaning, feature adding and metric measuring techniques, the BO algorithm was able to be cleanly and quickly constructed to work on any time-series data with minimal adjustment. The steps that were required for creating the pipeline were as follows:

- The data was pre-processed and prepared for feeding into the pipeline. Imputation was also performed in this step.
- Data was fed into the ML pipeline. The necessary parameters that had to be included alongside the data were:
    a. folder_path: File path linking to the dataset.
    b. set_name: The name of the dataset (For saving purposes).
    c. target: The name of the target column to fit to.
    d. trend_type: The type of seasonal trend observed (Generally additive or multiplicative).
    e. epd: Data entries per day.
    f. future: How many steps into the future (horizon) to predict for.
- Feature adding would be automatically conducted at this stage.
- The grid search would be created to find the optimal scaling and dimensionality reduction parameters. A low-dimensional NN model employing 5-fold cross-validation was used to determine the most effective cleaning parameters. These parameters would then be saved within a .csv file for future use.
- The fully cleaned data was then placed through the Bayesian Optimisation process. The optimal hyperparameters were then attained.
- Five-fold cross-validation was then used in forecasting future values for the given dataset. The predictions, along with metrics, were stored for future use.
- A web dashboard was integrated to display the results of training each ML model. These results could be cross compared with other models as well as a baseline for any dataset that had been trained on.

This process can be found in flowchart format in Figure 18.

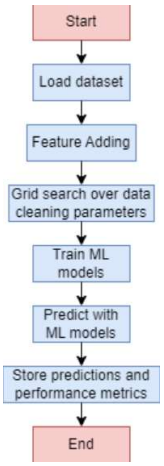


Figure 18: Flow chart for the BO algorithm

The Bayesian Optimisation algorithm was able to generate Random Forests and XGBoosts from the Sklearn library [40]. These were created using an inbuilt Bayesian Optimiser from the Scikit-Optimise library [41]. Table 3 includes a list of hyperparameters that could be adjusted using this method.

Table 3: Adjustable hyperparameters for random forest and XGBoost models using BO.

Model	Variable	Description	Values
Random Forest	max_depth	Maximum depth of the tree	Range(10, 1200)
	min_samples_leaf	Minimum number of samples required to be a leaf node	Range(0.001, 0.5)
	min_samples_split	Minimum number of samples required to split a node	Range(0.001, 1)
	n_estimators	Number of trees in the forest	Range(5, 5000)
	Max_features	Number of features to consider when looking for a split	[sqrt, log2, None]
XGBoost	learning_rate	Learning rate for gradient boosting algorithm	Range(0.01, 1)
	min_child_weight	Min weights needed in each child node for a split	Range(0, 10)
	max_depth	Maximum depth of trees	Range(1, 50)
	subsample	Subsample ratio from training set	Range(0.01, 1)
	colsample_bytree	Subsample ration for the columns used, for each tree	Range(0.01, 1)
	reg_lambda	L2 regularisation on weights	Range(1e-9, 1)
	reg_alpha	L1 regularisation on weights	Range(1e-9, 1)
	gamma	Minimum loss reduction required for any update to the tree	Range(1e-9, 0.5)
	n_estimators	Number of trees	Range(5, 5000)

The XGBoost and RF models were ultimately omitted due to long runtimes and model inaccuracy. As such, performances from these models have been removed from the discussion and results. However, they served as the building blocks for the GA implementation, meaning that they were repurposed for future implementations.

The BO algorithm was more commonly used for generating NN models. These were all implemented using Bayesian Optimisation helpers from the Keras-Tuner library [42]. Basic neural networks, LSTM and CNN architectures were able to be created using the TensorFlow library [43]. Table 4 shows the list of NN hyperparameters that could be adjusted using the BO algorithm.

Table 4: NN hyperparameters available for tuning using BO.

Variable	Description	Value
hp_activation	The activation function on each layer	[relu, tanh]
hp_learning_rate	The learning rate employed by the Adam Optimizer	Range[1e-4, 1e-2]
hp_dropout	The dropout rate on each consecutive dropout layer	Range[1e-3, 0.5]
hp_reg	The degree of l1 regularisation	Range[1e-4, 1e-2]
hp_max_neurons	Maximum number of neurons to exist within the architecture	Range[10, 5000]
hp_neuron_pct	The percentage of the maximum neurons that would appear on each layer	Range[1e-3, 1]
hp_layer_1	Number of units on the LSTM or CNN layer	Range[1, 100]

5.1.4.1 Web Dashboard

Simply generating the metrics for each model's performance gives users a poor intuition into how well the model is predicting on a data set. With this in mind, it was decided that a web dashboard should be developed to better represent the results of each model's performance. This dashboard was able to display the following characteristics for any ML model that performed on any dataset, and compare them against a baseline estimate:

- A line plot of the best model's predictions overlaid against the actual values it was predicting.
- An error distribution showing the Rayleigh Fit. This could be used to determine if overfitting was occurring.
- A radial plot displaying the R2, MAE, MAPE and MAE characteristics of the final model.

As previously mentioned, there are four categories which the user may select from. These categories control which dataset is being displayed, the ML model to predict on it, an optional baseline to compare against, and how far into the future the prediction runs for.

Implemented using the Dash library [44] and inspired by their app gallery's windspeed dashboard [45], this interface was deployed on the web and can be accessed via the Energy-Forecasting GitHub repository [46]. Figure 19 provides an example of the web dashboard showing performance results.

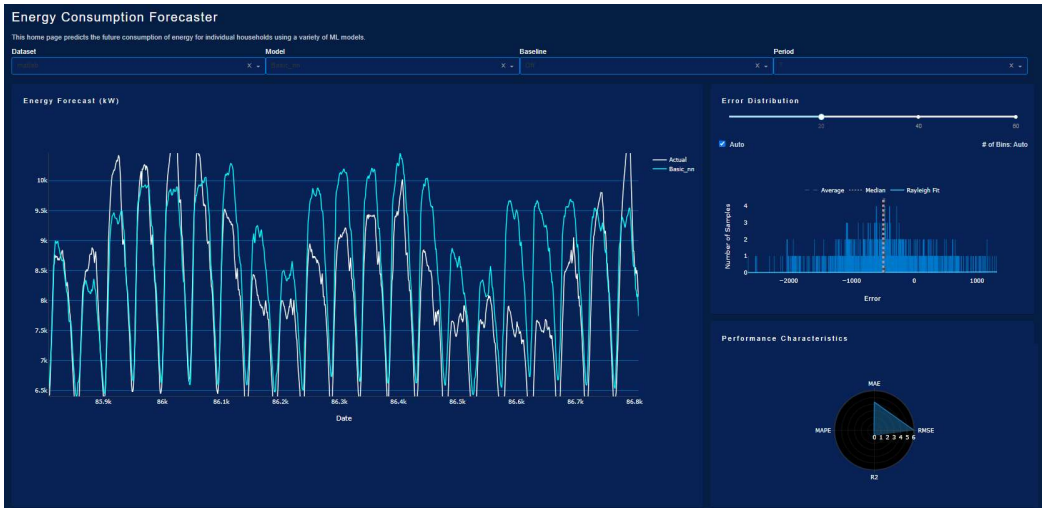


Figure 19: Online dashboard showing the performances of ML models on various datasets.

5.1.4.2 Findings from the BO and reasonings for GA Implementation

There were many limitations within the BO script that were realised upon the completion. These limitations included:

- Although the grid search for data cleaning was effective, its usefulness was limited. Aside from the long runtimes to finish, it was being tested on a model that differed from the final ones used in the BO process. As such, transferability of the grid search's optimum parameters could not be guaranteed.
- There existed no means of automatically tuning the window size. This had to be manually adjusted for each training sequence.
- The files were structured in such a way that was neither modular nor easy to use. Operating the BO algorithm required significant overhead in learning to use it, and as such was unfit for redistribution.
- Limited control was available over runtime characteristics. Users could not set max run times, save progress or load from previous runs.
- Keras requires significant computational power, meaning that the BO algorithms were not viable for standard PCs.

Although it was possible to fix some limitations, many of them could not be addressed as they would directly interfere with the Scikit-Optimize and Keras-Tuner libraries being used. Although there were obvious limitations with the model, there was no way to improve them with the way the architecture had been created. For this reason, a different approach was adopted. It was decided that a genetic algorithm would be created – manually, from scratch, that could satisfy these design problems and eventually supersede the BO algorithm.

5.1.5 Genetic Algorithm

The GA was created to address the flaws within the BO algorithm. It was designed with the fundamental intention of creating a fast, effective AutoML pipeline that could operate without the requirement of GPUs. The process of creating a genetic algorithm is far more hands-on, and required the construction of genomes, selection, cross-over and mutation functions.

5.1.5.1 Genome

All genetic algorithms work by constructing a population of randomly generated genomes. The genome within this algorithm would contain the parameters and hyperparameters required to create a time series forecaster. The genes within each genome, and what they represent, can be found within Figure 20 and Table 5:

```
genome_layout = ["window", "imputer", "outlier", "scaler", "dimension", "model"]
```

Figure 20: Structure of the Genome Layout

Table 5: List of genes that would be adjusted within each genome.

Parameter	Description	Values
Window size	Size of the forecasting window used	Range[1, 101]
Imputer	Type of value imputer used	[Mean, Median, Mode, None]
Outlier	Whether to remove values outside of 2.5 standard deviations	[True, False]
Scaler	Type of scaler to be used	[Standard, Normalizer, Robust, None]
Dimensionality Reduction	Type of dimensionality reduction or clustering to be used	[PCA, Feature Agglomeration, None]
Model	The ML model, along with its loaded hyperparameters	[Ridge, LassoLars, ElasticNet, Linear_SVR, Decision Tree, K-Nearest Neighbours, Random Forest, XGBoost]

The hyperparameters for generating Random Forests and XGBoost are identical to the BO implementation and can be found in Table 3. All other model implementations are currently experimental and not ready for deployment. From here, each genome within the population was nestled within a loadout dictionary containing all the parameter values, as well as its fitness score.

```
genome_dict = {"fitness": -math.inf, "genome": genome, "loadout": genome_loadout}
```

Figure 21: Genome dictionary created for every genome within the population.

5.1.5.2 Fitness Calculation

R2 was used as the metric for fitness calculation due to its ability to reflect a model's ability to accurately recreate a forecast [38]. Although MSE, MAE or MAPE could have been used, they were ultimately removed since a model's performance is inversely proportional to their scores in the above metrics. Furthermore, MSE introduces a heavy biasing towards outliers, and neither MSE nor MAE values will change drastically depending upon the dataset used [38]. R2 presented itself as the most natural metric for use under these conditions.

A K-fold cross-validation metric was used to remove any biasing, ensuring that a fair and accurate reflection of each genome's performance was returned. The fold value of cross-validation could be adjusted using the "cv" class variable.

5.1.5.3 Selection, Cross-Over and Mutation Functions

In line with the observations of Revello and McCartney [20], a selection function was designed that promoted elitism, whilst still allowing exploration of lower performing genomes. The selection function performed this using the following algorithm:

$$selection\_weight = 5 \times (20 - rank)$$

Where rank represents its relative rank in terms of fitness for that generation. This linear function was adopted because it was found to be more inclusive of lower performing models, whilst still selecting mostly high-achieving genomes. This helped inspire genetic diversity to a greater extent than exponentially weighted functions, which favoured the most successful genomes.

A number of parent genomes determined by the "n_successors" class variable were returned by the selection function to serve as the predecessors for each following generation. These parent genomes would then the cross-bred with each other to produce new offspring within the crossover function. This would repeat over all possible combinations of parent genomes returned by the selection function.

The crossover function would receive two parent genomes as an input, and return two child genomes, both of which had received an even distribution of genetics from the parent. Figure 22 represents the splitting process that was used to develop the child genomes.

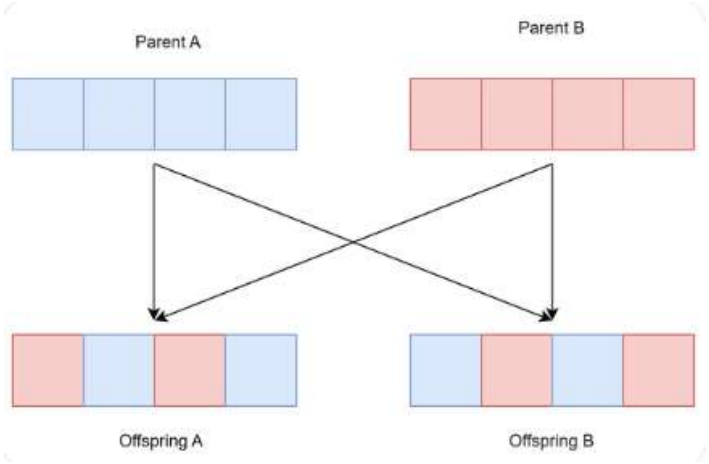


Figure 22: Representation of the genetic cross-over process.

Following the selection and cross-over functions, the next generation comprised of the following genomes:

- Parent genomes that were selected, with preference to the fittest genomes of the previous generation.
- Child genomes that were the product of genetic cross-over between the parent genomes.

To make up the remainder of the generation, a mutation function was used to randomly select from these genomes and return a mutated variant of them. The mutation function would perform a number of mutations determined by the "n_mutations" class variable, which would randomly perform the following actions upon the genome:

- Randomly change a hyperparameter value within the ML model.
- Randomly change an imputer, outlier removal, scaler or dimensionality reduction technique.
- Adjust the window size of the data input into the ML model.

This would be repeated until the number of newly generated genomes matched the predetermined size of each generation (Which could be adjusted using the "population" class variable).

5.1.5.4 The Over-All Process

From generating a population, to running fitness calculations and creating successive generations, the process undertaken by the GA can be found in Figure 23.

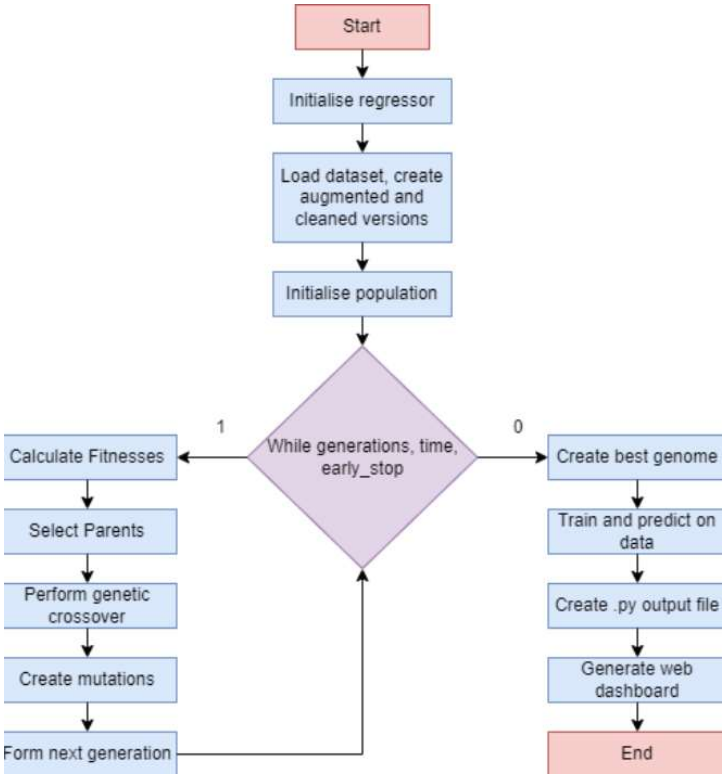


Figure 23: Flow Chart of the GA process.

5.1.5.5 Output Python File

An output .py file is created upon successfully running the evolutionary process employed by the GA. This .py file is the culmination of the entire AutoML pipeline, and can undertake the following steps in accordance with the optimum procedure found through evolution:

- Reading data in directly from the source.
- Data cleaning and feature adding.
- Applying scalers, dimensionality reduction and clustering.
- Generating an ML model, training and predicting on the given data.

This .py file was created with the foundational goal of achieving full automation of the entire pipeline. As such, every step detailed above is performed without user intervention, in the method best determined by the GA. Examples of these output files can be found within the appendices.

5.1.5.6 Web Dashboard

It was decided that simply generating performance metrics wouldn't capture the full capabilities of the GA. With this in mind, a simplified version of the web dashboard used alongside the BO algorithm was developed. This simplified web dashboard was able to display the following outputs at the end of the evolutionary process:

- A line plot of the best model's predictions overlaid against the actual values it was predicting.
- An error distribution showing the Rayleigh Fit. This could be used to determine if overfitting was occurring.
- A radial plot displaying the R2, MAE, MAPE, MAE and runtime characteristics of the final model.

An example of this dashboard can be found in Figure 24.

Figure 24: Simplified web dashboard created for GA.

5.1.5.7 Running the Time Series Forecaster

One of the biggest drawbacks of the BO algorithm was the messiness with which it was created. Although it was designed to be automated and accurate for data prediction, user-friendliness has suffered due to lack of attention. This user-friendliness became the most critical aspect of the GA, which was finally developed to be run with two simple functions: a class generation function and fitting function call. The TimeSeriesRegressor class can be initialised with the following function call.

```
class TimeSeriesRegressor:
    def __init__(self,
                 directory: str = os.getcwd(),
                 max_time_mins: int = math.inf,
                 mode: str = "normal",
                 crossover_rate: float = 0.1,
                 n_successors: int = 2,
                 checkpoint: bool = True,
                 warmstart: bool = False,
                 early_stop: float = 1,
                 cv: int = 5,
                 generations: int = 50,
                 population: int = 20,
                 n_jobs: int=1):

        """
        Initialise all parameters and inherent functions

        @param directory: Sets the directory for ml models
        @param max_time_mins: Sets the maximum runtime for the regressor
        @param mode: Determines which models are used
        @param crossover_rate: Rate at which successive genes are crossed over
        @param n_successors: Number of successful genomes that go into the next gen
        @param checkpoint: If enabled, saves the population every few iterations
        @param warmstart: If enabled, will resume from a previously saved checkpoint
        @param early_stop: Stops the process early if the MSE has been achieved
        @param cv: Number of cross-validations
        @param generations: Number of generations undertaken
        @param population: Size of each successive population
        """
```

After creating the TimeSeriesRegressor, it was necessary to run its fitting method on the desired dataset. This is achieved through the TimeSeriesRegressor's inbuilt 'fit' method, the documentation of which is as follows:

```
def fit(self,
        file_path: str,
        target: str,
        epd: int,
        trend_type: str = "None",
        future: int = 0
        ):

    """
    Fitting function to run evolution

    @param file_path: Path to the dataset
    @param target: Name of the data column to train for
    @param epd: Entries per day in the dataset
    @param trend_type: Trend type for seasonal decomposition
    @param future: Number of days to predict into the future
    """
```

6 Discussion and Results

Upon completion, both the GS/BO and GA algorithms were able to generate predictions for input datasets presented. However, given the goal of this thesis is to focus on electricity consumption, this will be the focus of the discussion and results. In testing, three electricity consumption datasets were used for 3 forecasting horizons – a total of 9 trials. As previously mentioned, the datasets used were as follows:

- AEMO's NEM dataset for NSW electricity demand.
- GEFCOM's 2012 electricity load dataset.
- Mathwork's 2016 dataset for electricity consumption.

Furthermore, the following forecasting horizons were applied for each trial:

- 1 step
- 1 day
- 1 week

Results were gathered from the GS/BO hybrid model employing basic NNs, LSTM and CNN architectures. These results were compared against the GA's performance, as well as a baseline persistence model.

6.1 AEMO NEM Dataset

Table 6: Results for 1 step forecast on NSW electrical demand sourced from AEMO's NEM dataset.

Algorithm	Model	R2	MAE	MAPE	RMSE	TIME (Hours)
Baseline	NF	0.996	43.707	0.009	57.380	N/A
GA	RF	0.993	44.966	0.011	58.730	0.743
BO	NN	0.379	435.544	0.109	573.95	6.450
	LSTM	0.356	430.520	0.100	584.668	10.660
	CNN	0.393	550.506	0.082	723.580	5.970
AEMO	AEMO	0.997	42.480	0.006	54.475	N/A

Table 7: Results for 1 day forecast on NSW electrical demand sourced from AEMO's NEM dataset.

Algorithm	Model	R2	MAE	MAPE	RMSE	(Hours)
Baseline	NF	0.493	463.174	0.098	675.911	N/A
GA	XGBoost	0.956	119.789	0.029	153.770	0.852
BO	NN	-0.133	776.230	0.117	988.179	9.960
	LSTM	0.065	711.110	0.111	897.430	20.080
	CNN	0.063	696.000	0.109	899.178	17.330
AEMO	AEMO	0.940	198.157	0.025	260.681	N/A

Table 8: Results for 1 week forecast on NSW electrical demand sourced from AEMO's NEM dataset.

Algorithm	Model	R2	MAE	MAPE	RMSE	TIME (Hours)
Baseline	NF	0.350	533.885	0.112	768.996	N/A
GA	RF	-4.021	1493.907	0.330	1639.260	0.858
BO	NN	-0.169	802.082	0.124	1004.598	12.166
	LSTM	0.009	760.189	0.117	925.097	22.820
	CNN	-0.125	788.960	0.122	985.347	23.660
AEMO	AEMO	0.858	291.824	0.037	407.066	N/A

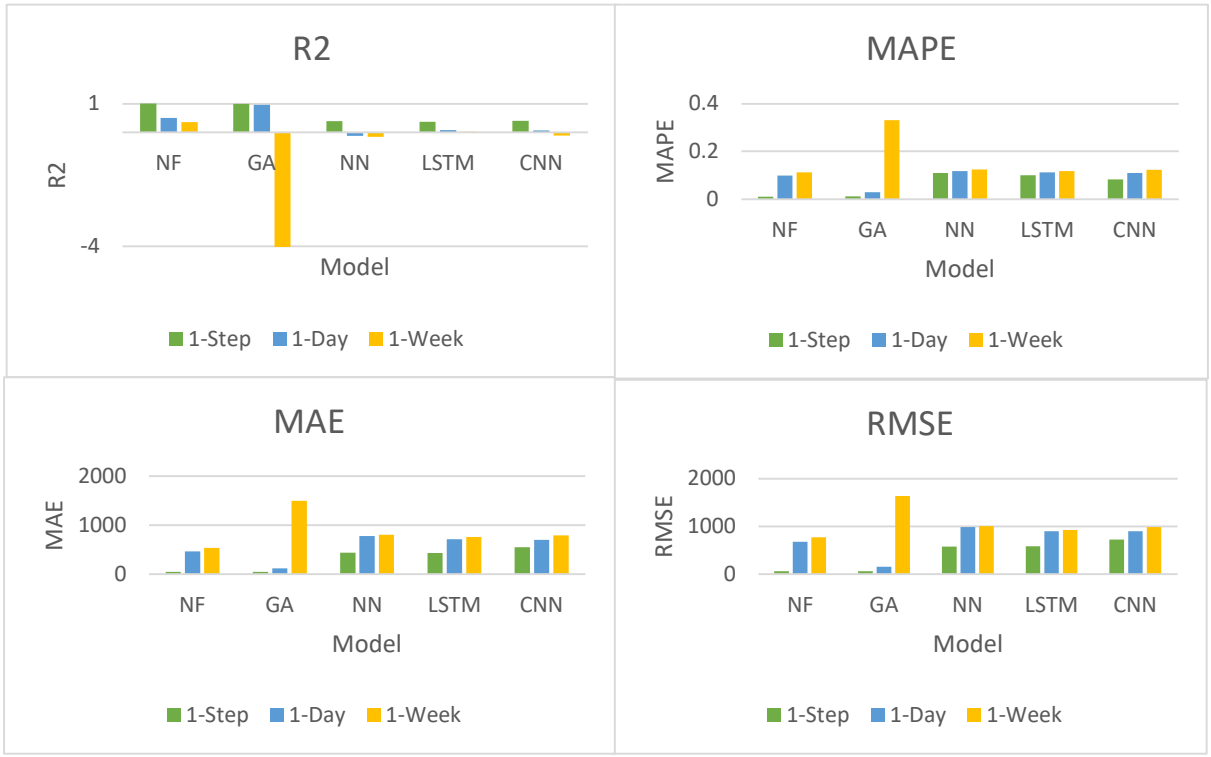


Figure 25: Comparison of metrics for different models on AEMO's NEM dataset.

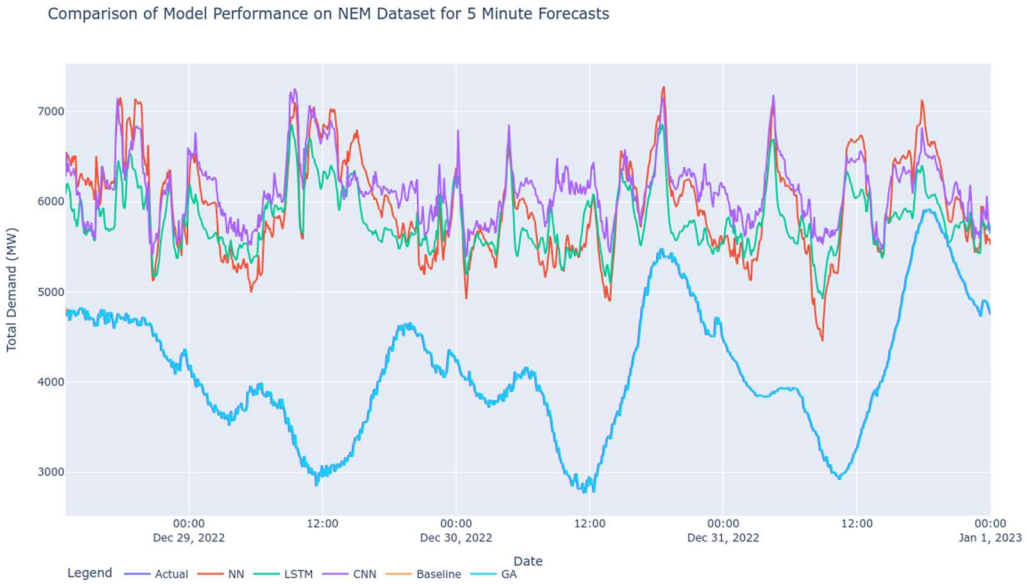


Figure 26: Comparison of 5-minute forecasts on AEMO's NEM dataset.

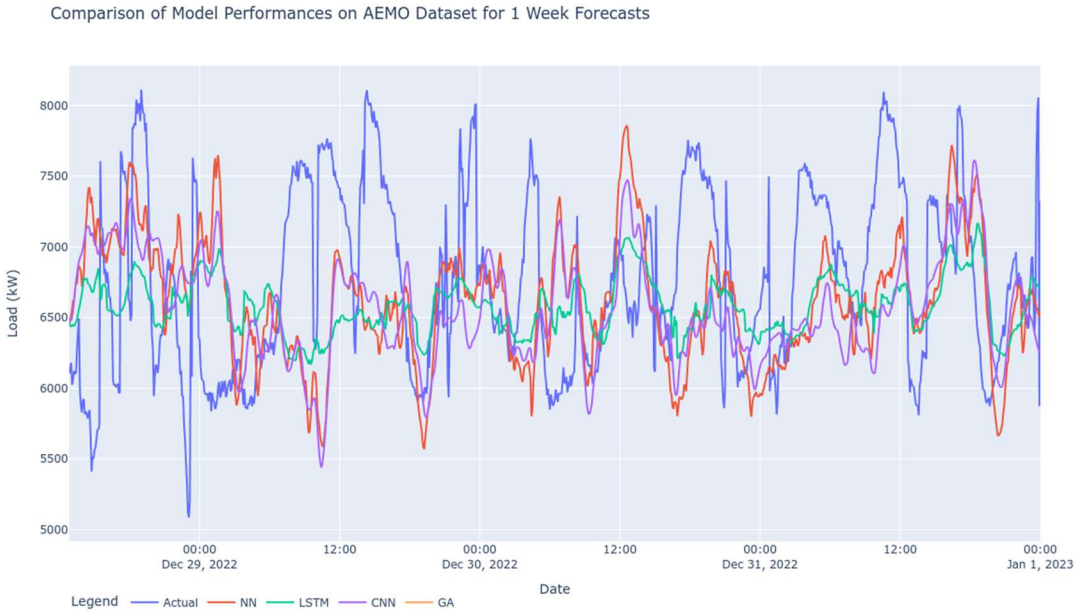


Figure 27: Comparison of 1-week forecasts on AEMO's NEM dataset

Figure 25 shows a graphical comparison of each model's performance metrics with respect to the increase in forecasting horizon. This provides a visual representation of the data found within Table 6, Table 7 and Table 8. To give a better intuition of how well each model predicted, Figure 26 and Figure 27 compare the output predictions for each model for 1-step and 1-week forecasts.

Several critical observations can be gained from testing on AEMO's NEM dataset. These observations, which will be explored in greater detail, are as follows:

- All GS/BO architecture models were unable to converge for any forecasting horizon.
- Although the GA performed the best of all algorithms for 1-step and 1-day forecasts, it had the poorest performance for 1-week horizons.
- All models feature a notable decrease in accuracy as the forecasting horizon increases.

As noted above, all GS/BO models were unable to converge for any given dataset. Analysis of the 5-minute (1-step) forecasts show that there is a significant offset for its predictions, thus indicating that severe underfitting has occurred. Although this offset disappears over larger forecasting horizons, each NN architecture lacks the ability to follow the trend of the total electricity demand.

This observation of model underfitting is fortified in analysing the GS/BO's performance metrics. The combination of 0.065 R2 and 11.1% MAPE for the LSTM architecture on 1-day forecasts indicate that the models have found no accurate method of converging their data. These results can be generalised to all GS/BO models over every forecasting horizon and point towards underfitting.

Contrary to the performance of the GS/BO architecture, the GA can perform with high accuracy over periods of 1-step and 1-day. With R2 values of 0.99 and 0.96 respectively, it outperforms all other generated algorithms, and rivals AEMO's forecasting accuracy. Over these periods, AEMO outperformed the GA's prediction by 0.4% and 0.5% MAPE.

Unfortunately, the GA is observed to have performed the poorest in long-term forecasts. With an R2 value of -4.021 for 1-week predictions, it has the lowest accuracy of all models for this period. This is a recurring theme between datasets and is examined in greater detail at the end of the discussion section.

The final observation, evident for each model across trials, is that its accuracy decreases along with the increase in forecasting horizon. Literature suggests [3, 5, 6] that, the further into the future a model attempts to predict, the more its accuracy will deteriorate. This decrease in accuracy is approximately linear for all models except the GA, which decreased sporadically at the 1-week prediction mark.

6.2 Mathworks Dataset

Table 9: Results for 1 step forecast on electrical consumption sourced from Mathwork's dataset.

Algorithm	Model	R2	MAE	MAPE	RMSE	TIME (Hours)
Baseline	NF	0.969	185.822	0.021	247.767	N/A
GA	RF	0.997	49.870	0.006	68.017	3.000
BO	NN	0.899	319.634	0.035	436.445	11.520
	LSTM	0.663	695.513	0.078	797.027	24.870
	CNN	0.886	343.673	0.038	463.444	19.180

Table 10: Results for 1 day forecast on electrical consumption sourced from Mathwork's dataset.

Algorithm	Model	R2	MAE	MAPE	RMSE	TIME (Hours)
Baseline	NF	0.763	459.960	0.051	684.625	N/A
GA	RF	0.997	51.300	0.005	70.419	3.000
BO	NN	0.735	543.590	0.062	706.961	3.910
	LSTM	0.861	365.684	0.040	510.503	9.703
	CNN	0.752	504.913	0.055	684.069	5.753

Table 11: Results for 1 week forecast on electrical consumption sourced from Mathwork's dataset.

Algorithm	Model	R2	MAE	MAPE	RMSE	TIME (Hours)
Baseline	NF	0.780	437.184	0.048	660.258	N/A
GA	XGBoost	0.983	128.876	0.014	180.690	3.666
BO	NN	0.772	503.937	0.059	655.041	4.636
	LSTM	0.796	457.000	0.051	620.426	7.180
	CNN	0.788	445.507	0.049	632.584	6.323

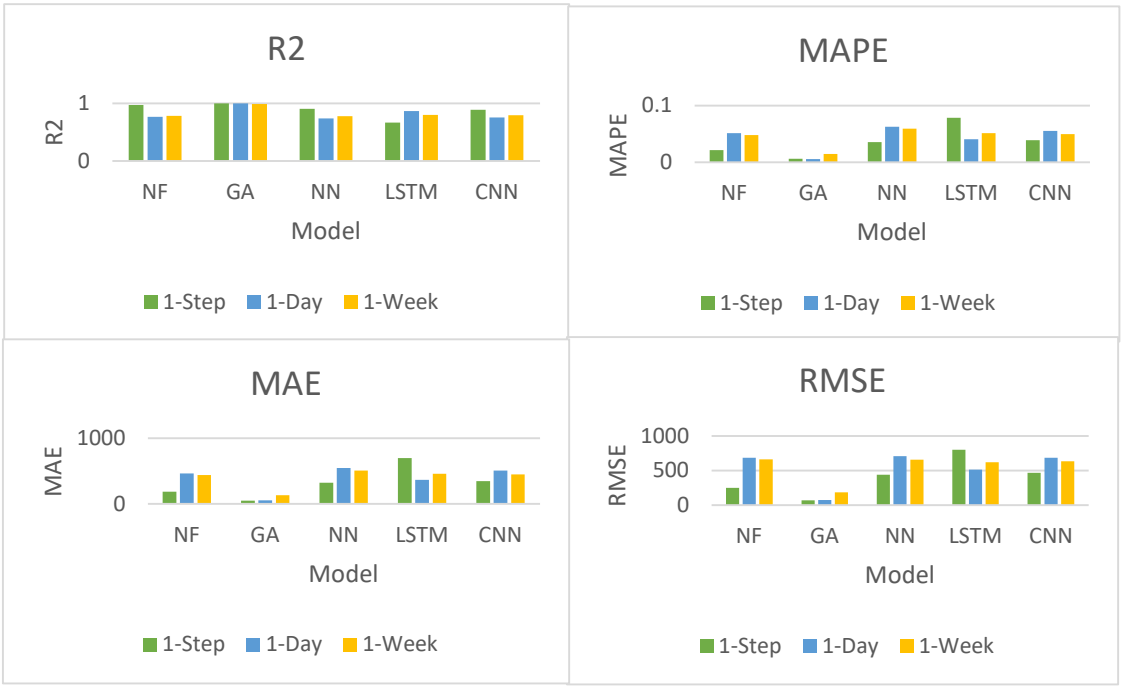


Figure 28: Metrics comparison for different models on the Mathworks dataset.

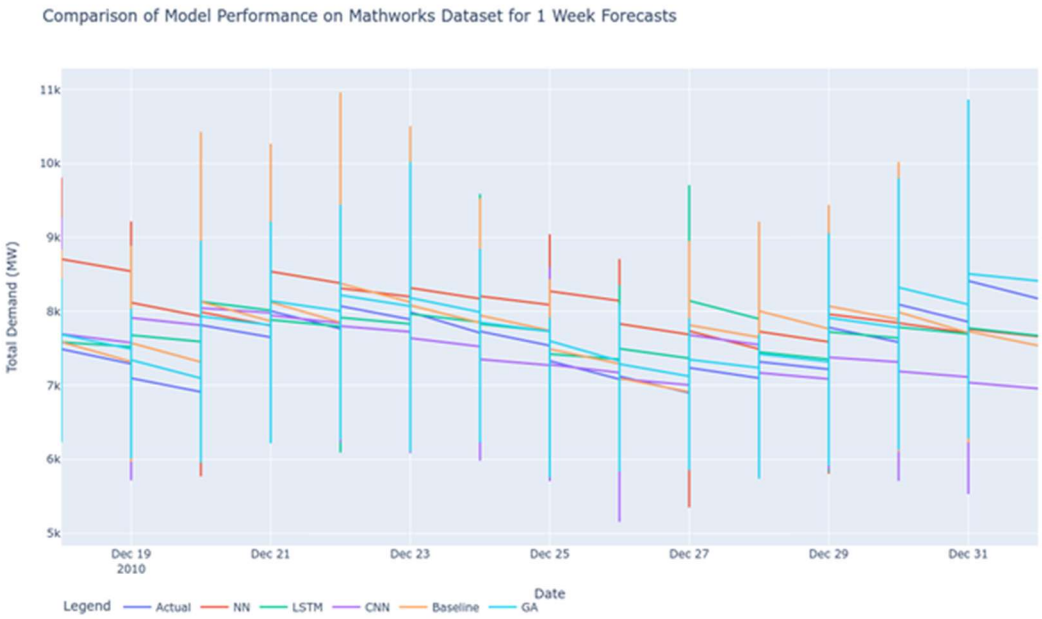


Figure 29: Comparison of 1-week forecasts on the Mathworks dataset.

The high-performance metrics of all models across the Mathworks dataset indicate that it is far easier to predict on than NEM data. Figure 29, which shows the model's predictions for 1 week ahead, indicates that all models were able to converge and predict accurately on the trend. That is not to say that all models performed equally, as the GA still performed strongest of all algorithms across the board.

Results gained from the Mathworks dataset serve to further validate the observations from NEM data. Additionally, far less underfitting has occurred for all GS/BO implementations than in the NEM dataset. In particular, Figure 28 shows that all GS/BO implementations return similar performance metrics. For these trials, they can perform approximately as accurately as the baseline persistence model. Of the NN architectures, the basic model performed the best despite requiring the shortest training times of 11.52, 3.91 and 4.63 hours across the three forecasting horizons.

All models, however, lagged considerably behind the GA. With R2 values of 0.997, 0.997 and 0.983 across the three forecasting horizons, it was able to predict almost perfectly in every test case. However, in analysing Figure 29, it becomes clear that all models were able to converge on the data's trendline. All models across every trial were able to consistently fit midnight fluctuations and fit a linear decline in consumption throughout the rest of the day. The GA was simply able to fit far more accurately. Its final predictions were able to run with 0.564%, 0.5% and 1.4% MAPE, indicating that it perfectly converged and predicted across all horizons. This was the only dataset out of the three in which the GA performed perfectly over a 1-week period, although this is mainly attributed to the dataset being easier to predict on (Verified by the notable higher performances of all models on the MathWorks data).

6.3 GEFCOM Dataset

Table 12: Results for 1 step forecast on electrical consumption sourced from GEFCOM's dataset.

Algorithm	Model	R2	MAE	MAPE	RMSE	TIME (Hours)
Baseline	NF	0.937	1131.393	0.062	1449.569	N/A
GA	GA	0.991	400.138	0.020	574.940	3.000
BO	NN	0.576	2730.965	0.1365	3759.064	3.600
	LSTM	0.682	2366.159	0.120	3253.834	7.770
	CNN	0.655	2505.060	0.129	3388.620	5.410

Table 13: Results for 1 day forecast on electrical consumption sourced from GEFCOM's dataset.

Algorithm	Model	R2	MAE	MAPE	RMSE	TIME (Hours)
Baseline	NF	0.507	2915.269	0.154	4072.138	N/A
GA	GA	0.983	472.705	0.024	668.690	3.000
BO	NN	0.522	2903.322	0.146	3992.227	3.457
	LSTM	0.503	3002.267	0.154	4070.907	6.800
	CNN	0.496	3049.085	0.158	4095.366	5.477

Table 14: Results for 1 week forecast on electrical consumption sourced from GEFCOM's dataset.

Algorithm	Model	R2	MAE	MAPE	RMSE	TIME (Hours)
Baseline	NF	0.193	3747.481	0.199	5199.794	N/A
GA	RF	0.746	2011.517	0.105	2617.632	3.300
BO	NN	0.365	3262.004	0.163	4574.370	3.860
	LSTM	0.352	3334.990	0.168	4625.113	8.620
	CNN	0.303	3343.813	0.159	4794.845	6.010

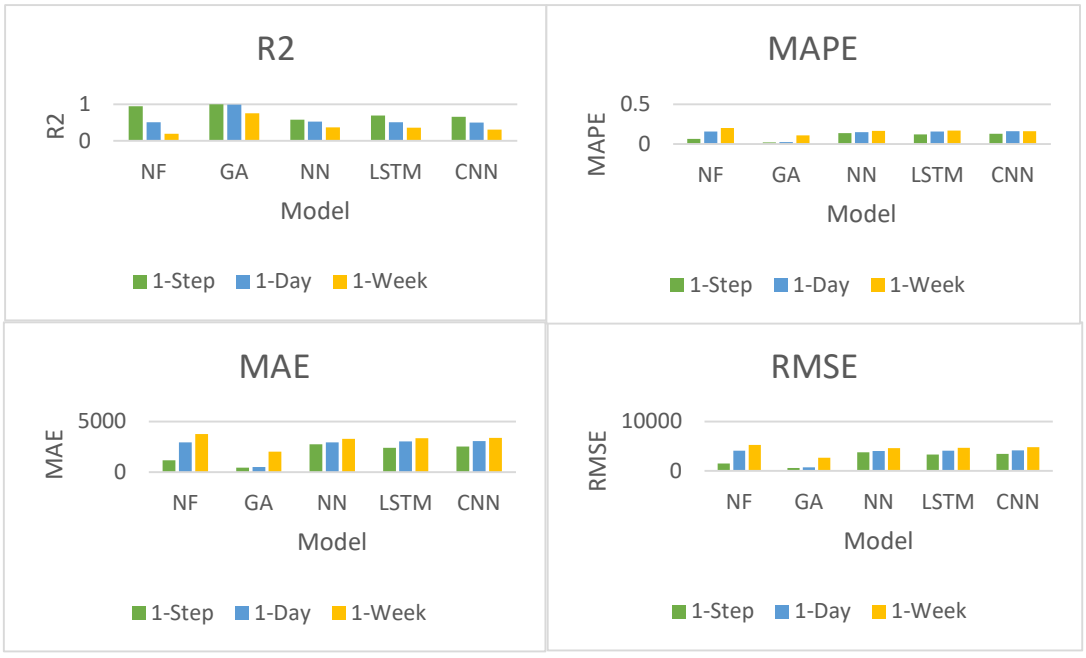


Figure 30: Metrics comparison for different models on the GEFCOM dataset

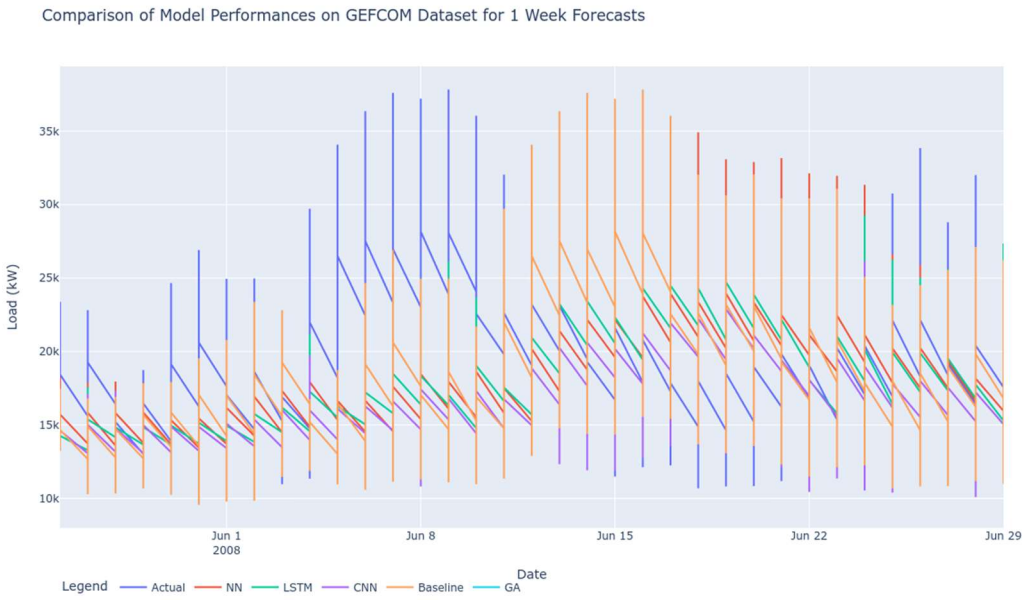


Figure 31: Comparison of 1-week forecasts on the GEFCOM dataset.

Similar observations from the first two datasets can be observed in the outputs from GEFCOM's 2012 Load forecasting data. Table 12, Table 13 and Table 14 show that the GS/BO models once again perform weakly, netting performance metrics that are – on average – comparable to the baseline model. With R2 values of 0.682, 0.503 and 0.352, the LSTM model performed the best on average out of the NN architectures. However, considering that it takes 115% longer to train on 1-step forecasts than the basic model, the 1.6% reduction in error isn't justifiable.

Once again, GA performs the best across the board on the GEFCOM Load data. However, a noticeable decrease in accuracy is observable between 1-day and 1-week forecasts. Although not as pronounced as that found on the AEMO dataset, its MAPE drops from 2.4% to 10.5% - a 437% increase in error. Figure 31 shows that all models experienced similar reductions in predictive capabilities over long-term forecasts.

The decrease in accuracy over a 1-week period is found to be universal for all ML models. Since this mirrors the behaviour observed in AEMO's dataset, this serves to highlight the variability of long-term prediction. This decrease in accuracy is not found within the MathWorks' dataset, although all models generated far greater performance metrics in those trials. This is more due to the simplicity of the Mathworks data than the model's predictive capabilities.

6.4 General Observations

Several observations could be generalised to performance across all trials. These observations are:

- The forecasting accuracy of all models is inversely proportional to the forecasting horizon. As the forecasting horizon increased, their predictive capabilities were reduced.
- All GS/BO models performed similarly. The basic NN most often performed with the least accuracy and shortest training time, while LSTM models generally had the highest accuracy with the longest train time. CNN models tended to take the Goldilocks zone between train time and accuracy.
- Severe underfitting has occurred in most GS/BO models.
- The baseline persistence model tended to predict as well as, if not better than, the GS/BO architectures. This was observed across 8 of the 9 trials.

Throughout the trials, the GS/BO architectures were observed to have the lowest performance metrics. Typically, even the baseline model would outperform them. This is likely due to two factors. The first of these factors, and most likely the main cause for inaccuracy, is linked to the lack of cohesion within the GS/BO's pipeline.

In creating the GS/BO pipeline, a lack of cohesion existed between the data cleaning and model tuning stages. Figure 32 shows a simplified representation of this pipeline. As can be observed, the data cleaning and model tuning steps are isolated from each other. Due to the nature of the Python packages used to implement this architecture, it was impossible to integrate data cleaning alongside model tuning. For this reason, data cleaning had to be performed before-hand.

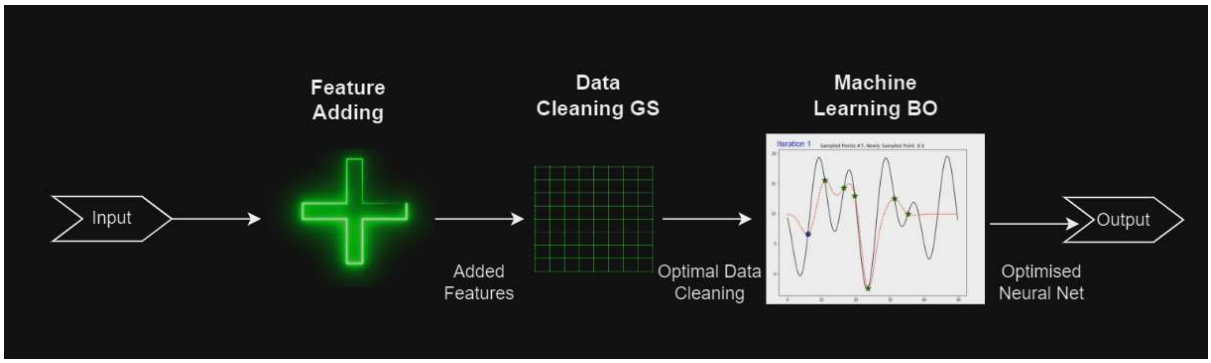


Figure 32: Simplified representation of the GS/BO architecture

To handle this problem, the GS over data cleaning parameters was tested using a linear regression model. The hope was that using the data cleaning parameters that performed the best for a simple model would generalise to optimum performance across all models.

This hypothesis was proven to be incorrect. Data cleaning, which is widely attributed to be among the most critical phases in ML pipelines [17], should be specifically tailored to each individual model [17]. That is to say, each individual model will behave differently to different methods of data cleaning. Attempting to generalise this step with the use of a linear regression model resulted in parameters that were optimised for the wrong model.

This lack of cohesion is likely the largest cause for inaccuracy within the GS/BO architecture. However, due to its reliance on external Python packages, this cannot be easily remedied. Keras-Tuner is an AutoML library that doesn't facilitate data cleaning [42], meaning that there is no quick fix for this model. It is recommended that future research does not attempt to recreate this method of automating.

The second cause for inaccuracy in the GS/BO pipeline relates back to the architecture of the NN models used. As recommended by Nikbakht et al [32], shallow NN models were favoured over those with deeper architectures to emphasize faster training time. This was paired with training and optimisation steps that were performed quickly rather than rigorously. Across all trials, the BO optimisation step was allowed to run for 20 epochs. Following this, the best performing model was trained on data for 200 epochs.

Given the complexity of the data that was being input, the number of layers, neurons, optimisation epochs and training epochs could have all been increased. This would result in a far greater training time, but would serve to reduce underfitting [18]. In contrast to the GS/BO pipeline, the GA could generally be considered to have performed strongly in most use cases.

Through thorough testing and analysis, it was found that the GA performed the best of all algorithms, across the board. Apart from forecasting on the NEM dataset for a 1-week horizon, the GA had the strongest performance metrics in every trial. Furthermore, in 77.78% of trials, the GA had a MAPE of between 0.5-2%. Furthermore, with a maximum run-time of 3 hours, it was able to converge far more rapidly than the GS/BO hybrid architecture. With these results, it can be concluded that the GA has satisfied the thesis' goal of creating a fast, accurate AutoML pipeline.

That being said, the GA is far from perfect. Although nigh perfect results were observed for 1-step and 1-day forecasts, its predictive capacity diminished rapidly with the increase in forecasting horizon. 1-week forecasts on the GEFCOM 2012 Load dataset returned R2 values of 0.746, while NEM predictions dropped to -4.021. These were accompanied by similarly diminished MAPE results of 10.5% and 33% respectively. This drop in predictive capability can be attributed to two main factors: data complexity and feature adding optimisation.

Perhaps the most prevalent cause of inaccuracy for week-long forecasts was the data complexity. All trials for the GA were conducted using a skeleton dataset containing only the load values it was attempting to forecast. Due to the heavy emphasis on feature adding, the algorithm was able to automatically generate a wide array of columns to add to the dataset. However, the fact remains that many columns of data were omitted to boost the running speed of the algorithm.

The NEM dataset, for example, initially had over 100 columns of data, and the Mathworks dataset with 9. Each of the columns contained important information pertaining to the weather and load conditions, many of which carry a strong correlation to the prediction target. Removing this information drastically increased the speed at which the GA was able to predict, at the cost of lowered accuracy. Future trials in which the GA is run with fully integrated datasets is recommended, as it is hypothesized that this will once again increase its forecasting accuracy.

The GA's forecasting accuracy may also have diminished due to a sub-optimal feature adding pipeline. According to the works of Arora et al [13] and Deoras [1], different feature adding factors become more relevant for larger forecasting windows. These works observed that weekly seasonal

decompositions correlated better than daily decompositions for forecasting horizons of 1-week or greater. The works of Deoras suggested that weather data holds a higher correlation than feature adding for long-term forecasts [1].

Both the issue of data complexity and sub-optimal feature adding can be solved automatically using the genetic algorithm. An additional step can be integrated into the model, where it automatically selects columns that have the highest correlation with the target column. This could be achieved through harnessing correlation and setting a threshold by which to remove under-correlated data [38]. This threshold could be automatically tweaked within the genome. In undertaking this approach, larger datasets of greater complexity could be used, while the GA automatically selects the data of most relevance depending upon the forecasting horizon. This is a recommendation for future research into this area.

7 Recommendations and Conclusion

Within this thesis the design and implementation of an AutoML pipeline were described. The motivation, goals and scope of this project were established. The key performances indicators, as outlined within the Project Outline and Scope, were that the final pipeline had to be:

- Fast and accurate.
- Able to work on any time-series data.
- Useable with any computer set-up.
- Customisable and user-friendly.

Two pipelines were created with this intention. These pipelines harnessed techniques derived from relevant literature including data cleaning, feature adding and automated machine learning.

The first of these implemented a hybrid GS/BO algorithm. This pipeline created various NN architectures that predicted with 11-13%, 5-7% and 11-15% MAPE across AEMO, Mathworks and GEFCOM datasets respectively. These architectures performed the worst in most trials, being often outperformed by the baseline persistence model. Long training times and heavy reliance on GPU usage further detracted from the applicability of the design. The combination of low accuracy, long training time, limited user-friendliness and need for GPUs warranted the creation of a new algorithm to supersede this implementation.

In identifying the flaws of the first pipeline, a GA was developed as a successor algorithm. The created GA used a plethora of ML models, all of which ran effectively without GPU usage. This algorithm maintained and improved upon the functionality of the GS/BO implementation, with several user-centric design improvements. These included code cleaning and implementation changes, greater versatility through parameter setting, output visualisations and automated script writing.

Asides from being more usable, the GA outperformed all other implementations in 88.89% of trials. The accuracy of the forecasts generated through this algorithm were staggering. In 7 of the 9 trials, the GA generated a model with R2 scores of 0.98-0.995 and MAPE values of 0.5%-2%. The MAPE across most trials was a fraction of the GS/BO implementation, with 1240% improvement in some instances. The GA regularly performed the best of all models by an outstanding margin.

The high performance of the GA was further outlined when compared against AEMO's NEM forecaster. For 2 of the 3 forecasting horizons, the GA was able to perform almost as well as AEMO's forecaster. AEMO's forecaster produced 0.4% and 0.5% less MAPE on the 1-step and 1-day forecasts respectively [8]. However, when considering that the GA converged within 3 hours without the use of GPU, this difference in error is acceptable.

That being said, accuracy diminished significantly as the forecasting horizon was extended to 1 week. Although the GA's predictive capabilities remained unchanged on the Mathworks dataset, its R2 values were reduced to -4.021 and 0.746 on the NEM and GEFCOM datasets respectively. Much of this error could be mitigated in future studies by customising long-term forecast feature adding [13] and introducing more parameters into the input data [18].

In conclusion, the aims of this thesis have been achieved, on schedule and in full. A fast, accurate and user-friendly AutoML pipeline has been developed that specialises in time-series forecasting. The final algorithm was able to outperform all other models used within this research, although further study is needed to refine it. Numerous recommendations and directions for following work have been outlined within the Recommendations for Further Improvement.

7.1 Recommendations for Further Improvement

The Discussion and Results section of this report have indicated that the GA was able to outperform all other models in 88.89% of test cases. It was created with the intention of succeeding its GS / BO predecessor, and has succeeded it marked improvements in accuracy, training time and useability. That being said, there are many areas of potential improvement. Mostly concerned with the versatility of the design, these recommendations for future improvement include:

- Further trials should be run on the GA for 1-week forecasts using more input data, such as weather indicators. This should serve to mitigate the reduction in accuracy over long-term forecasts.
- The feature adding pipeline should be adjusted to better facilitate long-term forecasting. As noted by Deoras et al [1], unique feature adding processes should be implemented for short-term and long-term forecasting. This will help increase the accuracy of 1-week forecasts.
- Although RF and XGBoost models are some of the most robust and versatile ML models, they recent research into deep learning implementations have produced astounding results [6, 32, 33, 36]. The implementation of an evolutionary neural network could greatly enhance the GA's performance. Using a deeply layered NN may also remedy problems with long-term forecasts [6].
- Code modifications have introduced several bugs into the 'warm-starting' of the GA from previous training process. Until these are addressed, warm-start functionality cannot be guaranteed.
- RF and XGBoost models have been tested extensively throughout this thesis. KNN, SVR, Ridge, LassoLars and ElasticNet models have also been implemented, but without the same level of extensive testing. Further trials are recommended on these models to ensure their functionality.
- Pytest files are outdated and should be fixed.
- Training time regularly exceeds the maximum running time by a small margin. Further tests should be conducted to isolate this error.
- Tweaks within the GA's selection, split crossover and mutation functions could further enhance its optimisation.

Improvements for the BO algorithm are unnecessary, as it has already been deprecated following the GA implementation.

8 Appendices

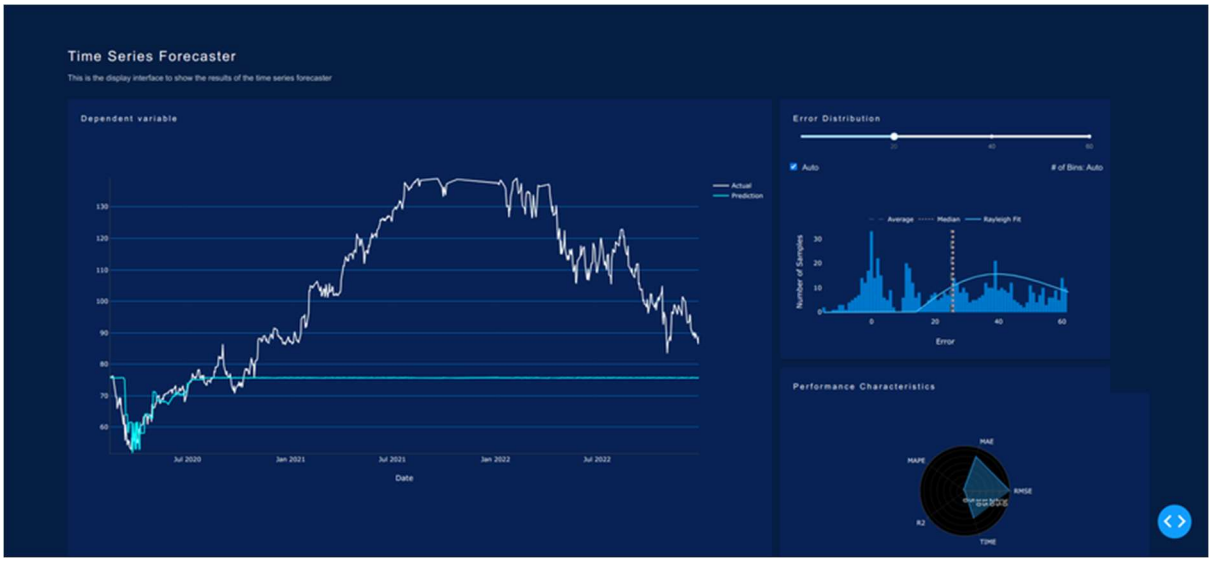


Figure 33: Example output of the GA's visualiser following 20 minutes runtime

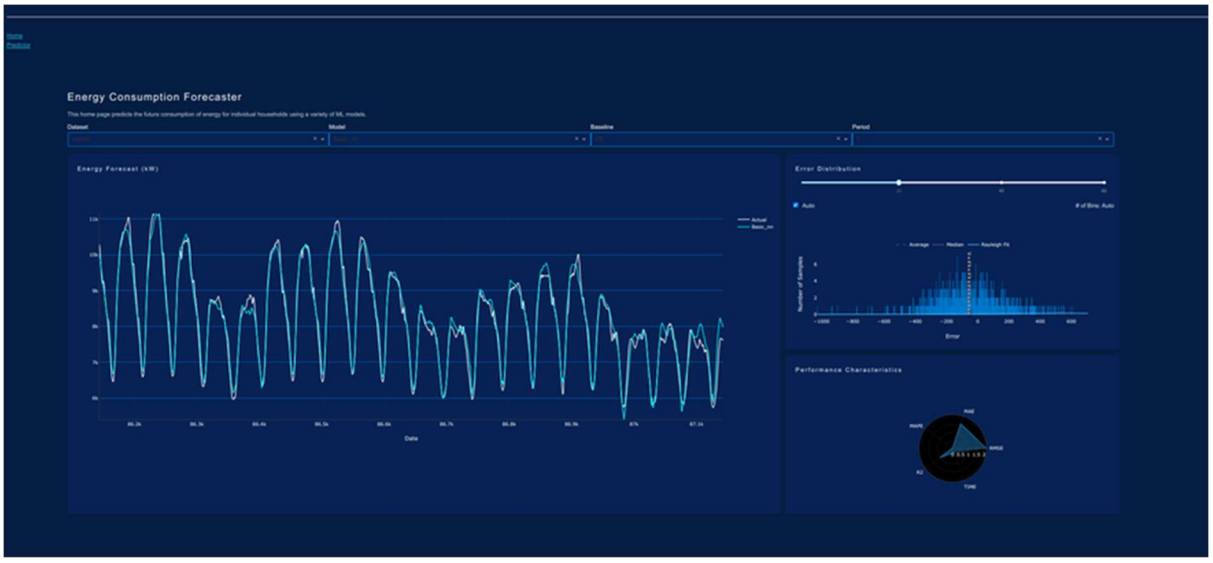


Figure 34: Example output from the GS/BO results visualisation

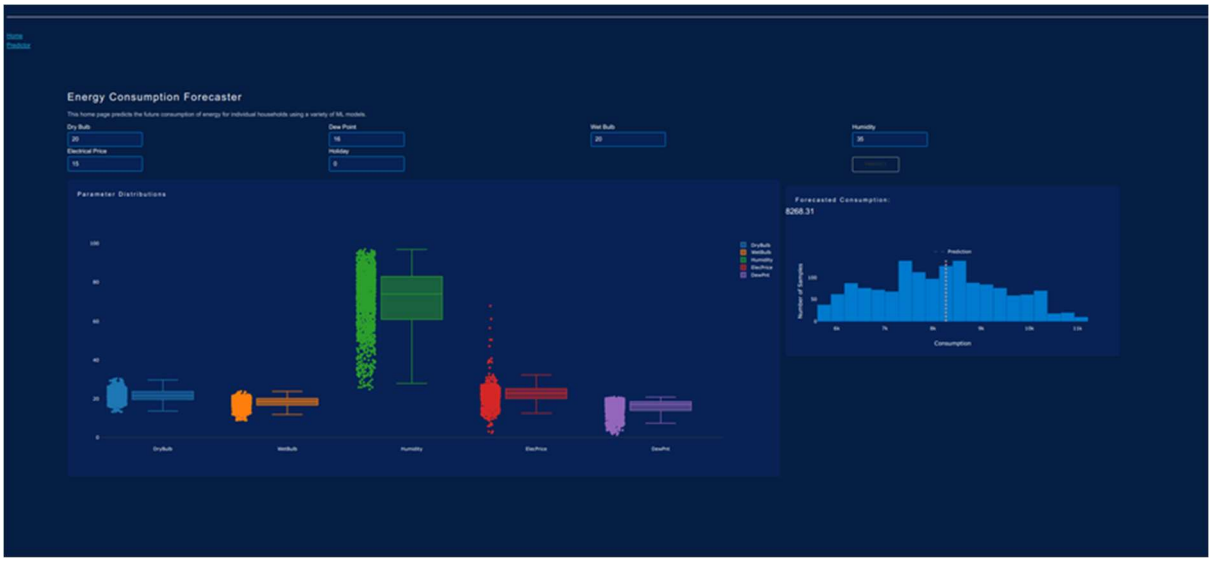
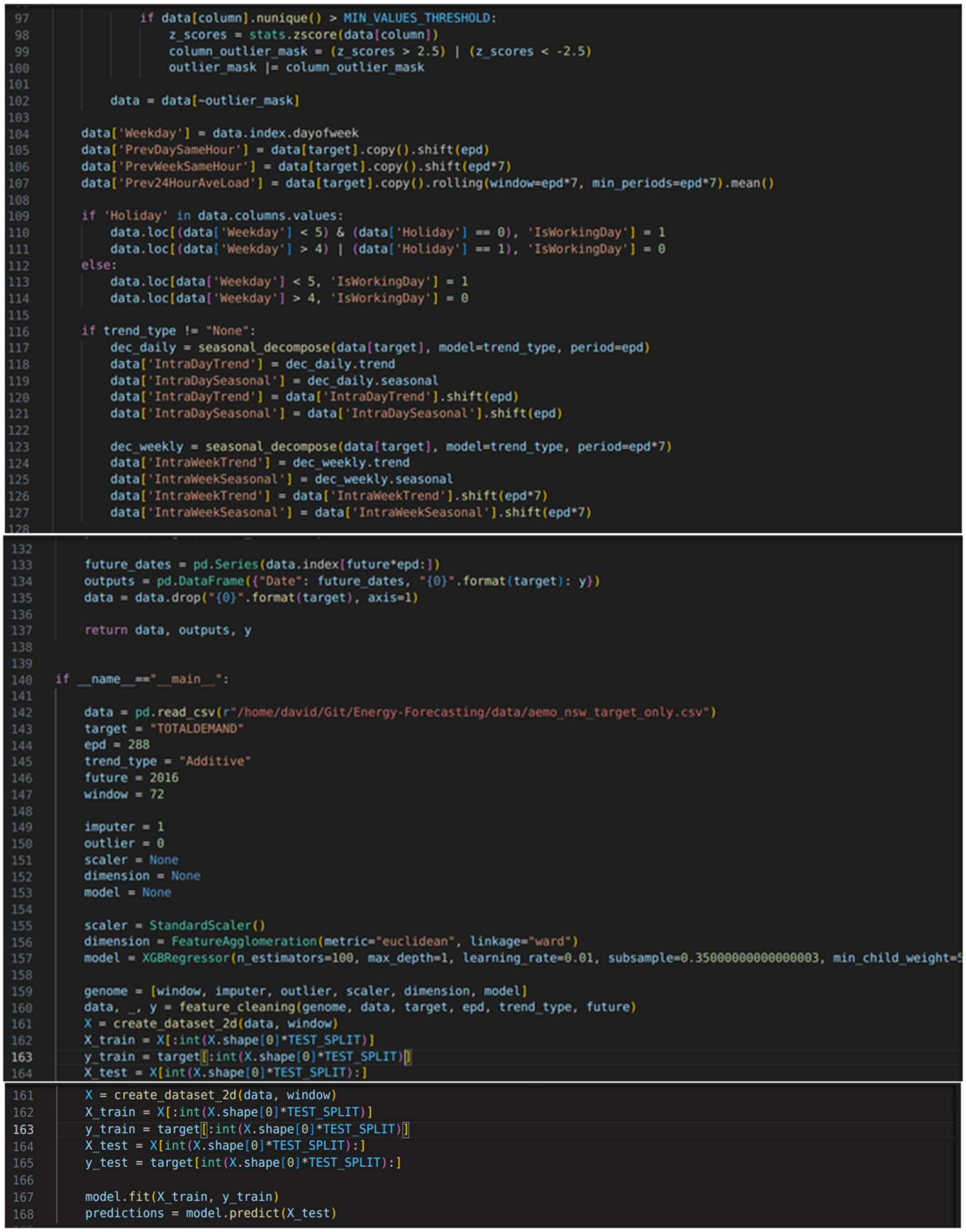


Figure 35: In-built electricity consumption predictor from the GS/BO implementation

```python
import pandas as pd
import numpy as np
import time
import math
import os
import logging
import json
from typing import List, Optional, Callable, Tuple
from random import randint, randrange, random, choices
from sklearn.linear_model import RidgeCV, LassoLarsCV, ElasticNetCV, Ridge, LassoLars
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import RobustScaler, Normalizer, StandardScaler, LabelEncoder
from sklearn.cluster import FeatureAgglomeration
from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.svm import LinearSVR
from sklearn.ensemble import RandomForestRegressor
from xgboost import XGBRegressor
from sklearn.decomposition import PCA
from sklearn.metrics import mean_squared_error as mse
from statsmodels.tsa.seasonal import seasonal_decompose
from scipy import stats


genome_layout = ["window", "imputer", "outlier", "scaler", "dimension", "model"]
TEST_SPLIT = 0.8
COLUMN_REMOVE = 0.3
MIN_VALUES_THRESHOLD = 100


def collapse_columns(data):

    data = data.copy()
    if isinstance(data.columns, pd.MultiIndex):
        data.columns = data.columns.to_series().apply(lambda x: "__".join(x))
    return data


def create_dataset_2d(input, win_size):

    np_data = np.array(input.copy())
    X = []

    for i in range(len(np_data)-win_size):
        row = [r for r in np_data[i:i+win_size]]
        X.append(row)

    X = np.array(X)
    X = X.reshape(X.shape[0], -1)

    return X


def feature_cleaning(genome, data, target, epd, trend_type, future):

    if epd <= 0:
        raise ValueError("Please input a positive integer for the entries per day")
    data = collapse_columns(data)

    col = None
    date_times = 0

    duplicate_columns = data.columns[data.columns.duplicated()]
    data.drop(columns=duplicate_columns, inplace=True)
    missing_percentage = (data.isna().sum() / len(data)) * 100
    columns_with_missing = missing_percentage[missing_percentage >= COLUMN_REMOVE].index
    data.drop(columns=columns_with_missing, inplace=True)

    for column in data.columns:
        if pd.api.types.is_datetime64_any_dtype(data[column]):
            col = column
            date_times += 1

    if 0 < date_times < 2:
        data.dropna(subset=[target, col], inplace=True)
        data = data.set_index(col)

    else:
        raise ValueError("Ensure that there is one and only one datetime column present within the dataset")

    imputer_val = genome[genome_layout.index("imputer")]
    outlier = genome[genome_layout.index("outlier")]

    if imputer_val > 0:
        if imputer_val == 1:
            imputer = SimpleImputer(strategy="mean")
        elif imputer_val == 2:
            imputer = SimpleImputer(strategy="median")
        if imputer_val == 3:
            imputer = SimpleImputer(strategy="most_frequent")
        ind = data.index
        data = pd.DataFrame(imputer.fit_transform(data), columns = data.columns).set_index(ind)

    if outlier:
        outlier_mask = pd.Series(False, index=data.index)
        for column in data.columns:
```



```
            if data[column].nunique() > MIN_VALUES_THRESHOLD:
                z_scores = stats.zscore(data[column])
                column_outlier_mask = (z_scores > 2.5) | (z_scores < -2.5)
                outlier_mask |= column_outlier_mask

        data = data[~outlier_mask]

    data['Weekday'] = data.index.dayofweek
    data['PrevDaySameHour'] = data[target].copy().shift(epd)
    data['PrevWeekSameHour'] = data[target].copy().shift(epd*7)
    data['Prev24HourAveLoad'] = data[target].copy().rolling(window=epd*7, min_periods=epd*7).mean()

    if 'Holiday' in data.columns.values:
        data.loc[(data['Weekday'] < 5) & (data['Holiday'] == 0), 'IsWorkingDay'] = 1
        data.loc[(data['Weekday'] > 4) | (data['Holiday'] == 1), 'IsWorkingDay'] = 0
    else:
        data.loc[data['Weekday'] < 5, 'IsWorkingDay'] = 1
        data.loc[data['Weekday'] > 4, 'IsWorkingDay'] = 0

    if trend_type != "None":
        dec_daily = seasonal_decompose(data[target], model=trend_type, period=epd)
        data['IntraDayTrend'] = dec_daily.trend
        data['IntraDaySeasonal'] = dec_daily.seasonal
        data['IntraDayTrend'] = data['IntraDayTrend'].shift(epd)
        data['IntraDaySeasonal'] = data['IntraDaySeasonal'].shift(epd)

        dec_weekly = seasonal_decompose(data[target], model=trend_type, period=epd*7)
        data['IntraWeekTrend'] = dec_weekly.trend
        data['IntraWeekSeasonal'] = dec_weekly.seasonal
        data['IntraWeekTrend'] = data['IntraWeekTrend'].shift(epd*7)
        data['IntraWeekSeasonal'] = data['IntraWeekSeasonal'].shift(epd*7)

    future_dates = pd.Series(data.index[future*epd:])
    outputs = pd.DataFrame({"Date": future_dates, "{0}".format(target): y})
    data = data.drop("{0}".format(target), axis=1)

    return data, outputs, y


if __name__=="__main__":

    data = pd.read_csv(r"/home/david/Git/Energy-Forecasting/data/aemo_nsw_target_only.csv")
    target = "TOTALDEMAND"
    epd = 288
    trend_type = "Additive"
    future = 2016
    window = 72

    imputer = 1
    outlier = 0
    scaler = None
    dimension = None
    model = None

    scaler = StandardScaler()
    dimension = FeatureAgglomeration(metric="euclidean", linkage="ward")
    model = XGBRegressor(n_estimators=100, max_depth=1, learning_rate=0.01, subsample=0.3500000000000003, min_child_weight=5

    genome = [window, imputer, outlier, scaler, dimension, model]
    data, _, y = feature_cleaning(genome, data, target, epd, trend_type, future)
    X = create_dataset_2d(data, window)
    X_train = X[:int(X.shape[0]*TEST_SPLIT)]
    y_train = target[:int(X.shape[0]*TEST_SPLIT)]
    X_test = X[int(X.shape[0]*TEST_SPLIT):]
    X = create_dataset_2d(data, window)
    X_train = X[:int(X.shape[0]*TEST_SPLIT)]
    y_train = target[:int(X.shape[0]*TEST_SPLIT)]
    X_test = X[int(X.shape[0]*TEST_SPLIT):]
    y_test = target[int(X.shape[0]*TEST_SPLIT):]

    model.fit(X_train, y_train)
    predictions = model.predict(X_test)
```

Figure 36: An example auto-generated .py file made from the GA's most effective genome

Figure 37: 1-day trend across the GEFCOM load data

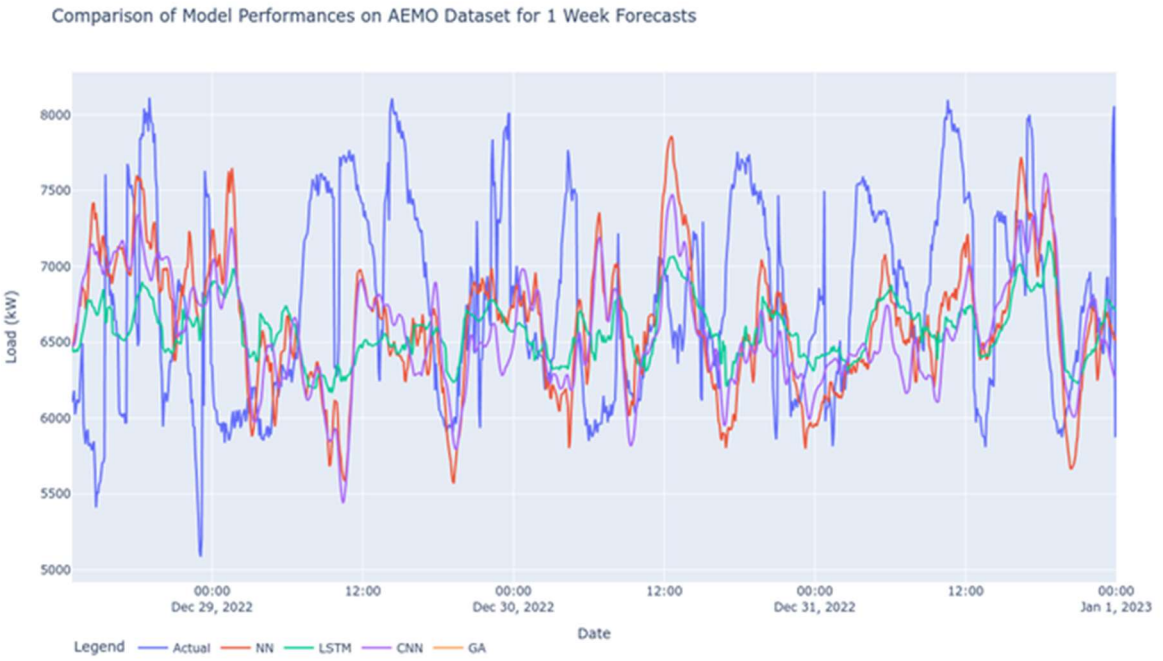


Figure 38: Comparison of 1-week forecasts on the AEMO dataset

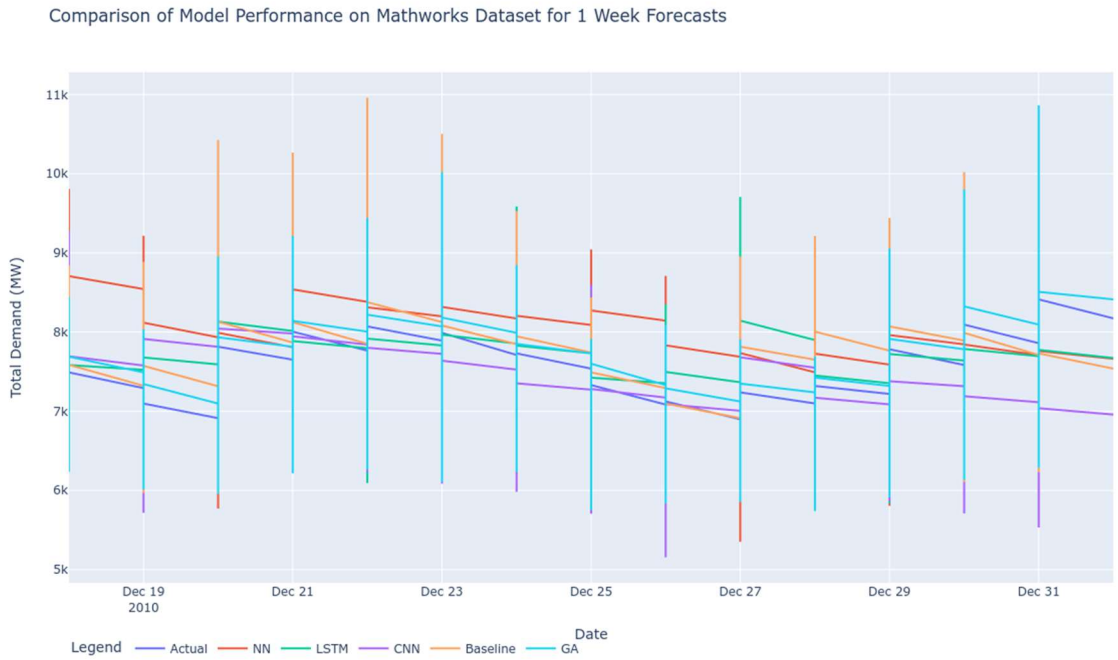


Figure 39: Comparison of 1-week forecasts on the Mathwork's dataset

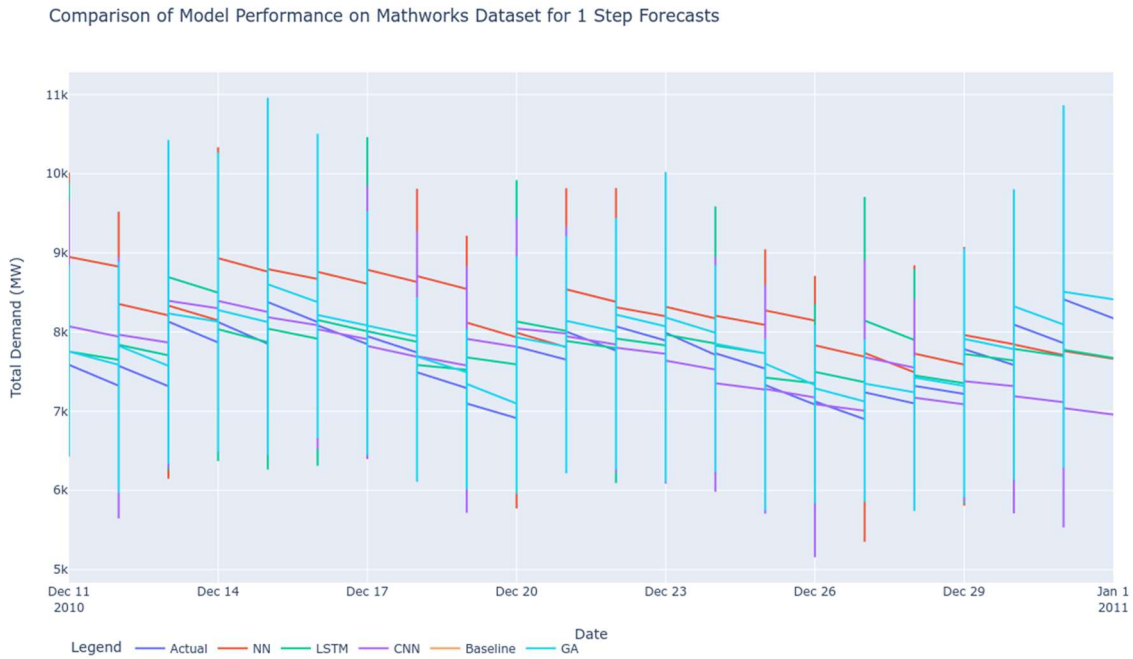


Figure 40: Comparison of 1-step forecasts on the Mathwork's dataset

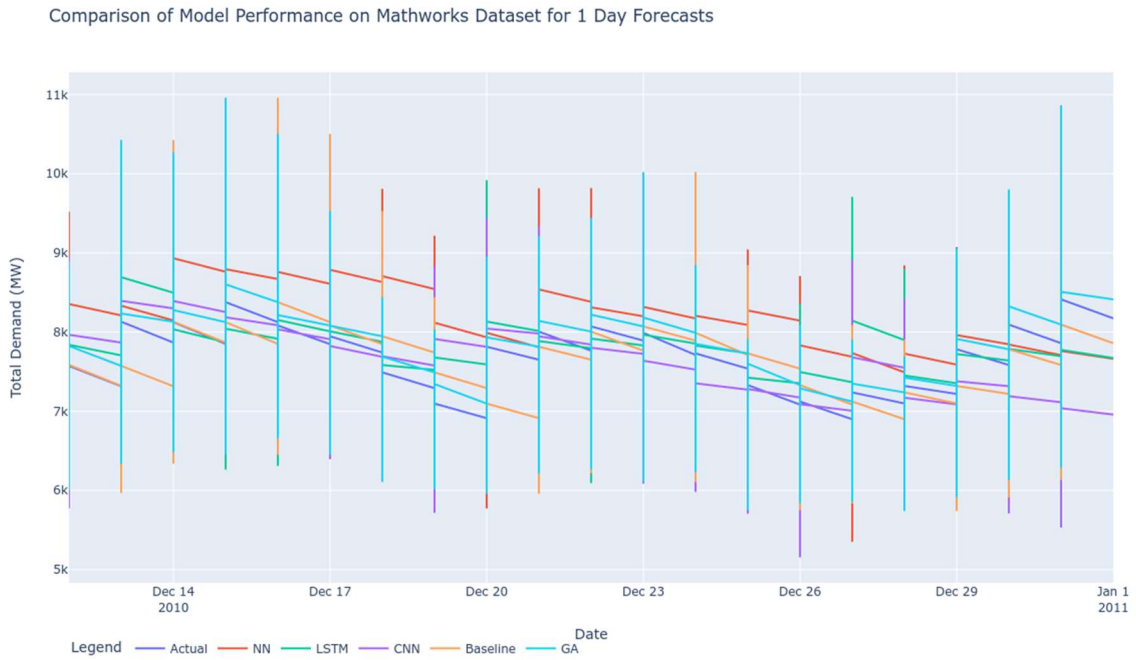


Figure 41: Comparison of 1-day forecasts on the Mathworks dataset

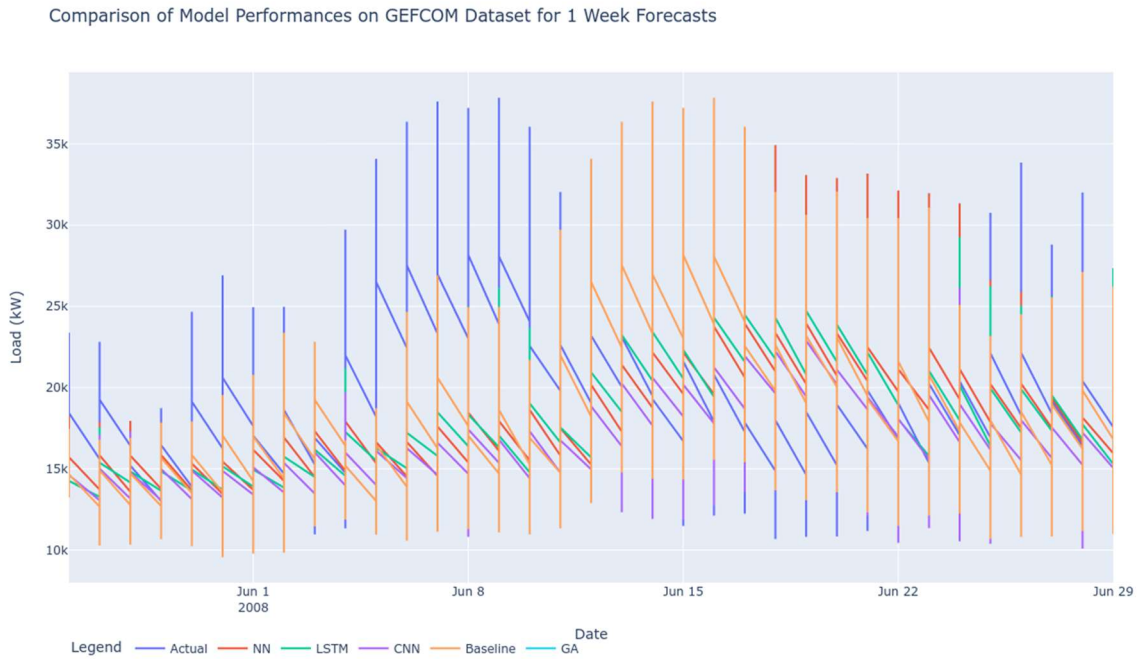


Figure 42: Comparison of 1-week forecasts on the GEFCOM load dataset

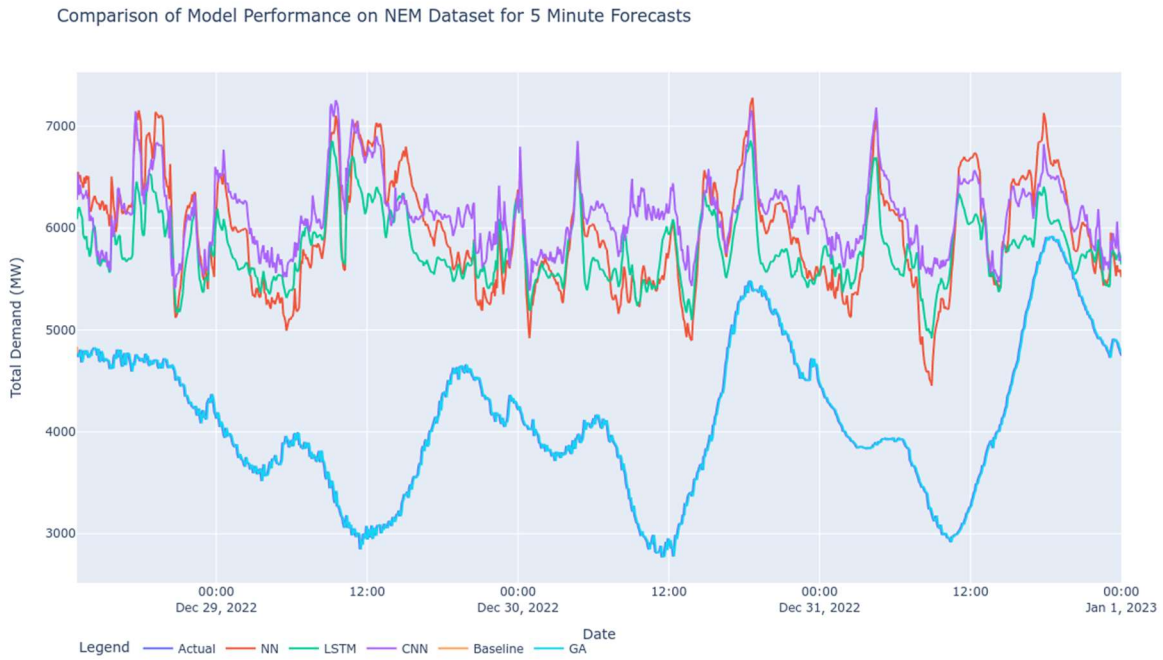


Figure 43: Comparison of 5-minute forecasts on the NEM dataset

9 References

[1] A. Deoras. "Electricity Load and Price Forecasting Webinar Case Study." Mathworks. https://au.mathworks.com/matlabcentral/fileexchange/28684-electricity-load-and-price-forecasting-webinar-case-study (accessed 30-08, 2023).

[2] T. Hong, P. Pinson, and S. Fan, "Global Energy Forecasting Competition 2012," *International Journal of Forecasting,* vol. 30, no. 2, pp. 357-363, 2014/04/01/ 2014, doi: https://doi.org/10.1016/j.ijforecast.2013.07.001.

[3] A. Alsharef, K. Aggarwal, Sonia, M. Kumar, and A. Mishra, "Review of ML and AutoML Solutions to Forecast Time-Series Data," *Archives of Computational Methods in Engineering,* vol. 29, no. 7, pp. 5297-5311, 2022, doi: 10.1007/s11831-022-09765-0.

[4] C. N. Babu and B. E. Reddy, "A moving-average filter based hybrid ARIMA–ANN model for forecasting time series data," *Applied Soft Computing,* vol. 23, pp. 27-38, 2014/10/01/ 2014, doi: https://doi.org/10.1016/j.asoc.2014.05.028.

[5] E. Bas, U. Yolcu, and E. Egrioglu, "Picture fuzzy regression functions approach for financial time series based on ridge regression and genetic algorithm," *Journal of Computational and Applied Mathematics,* vol. 370, p. 112656, 2020/05/15/ 2020, doi: https://doi.org/10.1016/j.cam.2019.112656.

[6] A. Azadeh, S. F. Ghaderi, S. Tarverdian, and M. Saberi, "Integration of artificial neural networks and genetic algorithm to predict electrical energy consumption," *Applied Mathematics and Computation,* vol. 186, no. 2, pp. 1731-1741, 2007/03/15/ 2007, doi: https://doi.org/10.1016/j.amc.2006.08.093.

[7] J. Waring, C. Lindvall, and R. Umeton, "Automated machine learning: Review of the state-of-the-art and opportunities for healthcare," *Artificial Intelligence in Medicine,* vol. 104, p. 101822, 2020/04/01/ 2020, doi: https://doi.org/10.1016/j.artmed.2020.101822.

[8] AEMO. "AEMO NEM Data " https://aemo.com.au/en/energy-systems/electricity/national-electricity-market-nem/data-nem (accessed 5/10, 2023).

[9] M. Kerai, "Smart Meter Statistics in Great Britain: Quarterly Report to end," Government of the United Kingdom, London, 2022.

[10] "AEMC sets out metering options for a smarter energy future," in *Australian Energy Marketing Commision*, ed, 2021.

[11] J. T. Siddharth Arora, "Forecasting electricity smart meter data using conditional kernel," *Elsevier,* pp. 47-48, 2014.

[12] "Review of the Regulatory Framework for Metering Services," Australian Energy Market Commision, Canberra, 2021.

[13] S. Arora and J. W. Taylor, "Forecasting electricity smart meter data using conditional kernel density estimation," *Omega,* vol. 59, pp. 47-59, 2016-03-01 2016, doi: 10.1016/j.omega.2014.08.008.

[14] J. W. Yao Zhang, "K-nearest neighbors and a kernel density estimator for," *Elsevier,* pp. 1075-1079, 2016.

[15] F. Nielsen, "Hierarchical Clustering," Springer International Publishing, 2016, pp. 195-211.

[16] E. Regnier, "Doing something about the weather," *Omega,* vol. 36, pp. 22-32, 02/01 2008, doi: 10.1016/j.omega.2005.07.011.

[17] H. Ma, "Thesis Discussions," D. Gaul, Ed., ed, 2023.

[18] S. Loussaief and A. Abdelkrim, "Convolutional Neural Network Hyper-Parameters Optimization based on Genetic Algorithms," *International Journal of Advanced Computer Science and Applications,* vol. 9, 01/01 2018, doi: 10.14569/IJACSA.2018.091031.

[19] P. Liashchynskyi and P. Liashchynskyi, *Grid Search, Random Search, Genetic Algorithm: A Big Comparison for NAS*. 2019.

[20] T. E. Revello and R. McCartney, "Generating war game strategies using a genetic algorithm," in *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No.02TH8600)*, 12-17 May 2002 2002, vol. 2, pp. 1086-1091 vol.2, doi: 10.1109/CEC.2002.1004394.

[21] T. Yan, A. Zhou, and S.-L. Shen, "Prediction of long-term water quality using machine learning enhanced by Bayesian optimisation," *Environmental Pollution,* vol. 318, p. 120870, 2023/02/01/ 2023, doi: https://doi.org/10.1016/j.envpol.2022.120870.

[22] G. W. Colopy, M. A. F. Pimentel, S. J. Roberts, and D. A. Clifton, "Bayesian optimisation of Gaussian processes for identifying the deteriorating patient," in *2017 IEEE EMBS International Conference on Biomedical & Health Informatics (BHI)*, 16-19 Feb. 2017 2017, pp. 85-88, doi: 10.1109/BHI.2017.7897211.

[23] C. L. Jonathon Waring, Renato Umeton, "Automated machine learning: Review of the state-of-the-art and opportunities for healthcare," *Elsevier,* p. 104, 2020.

[24] M. Akil, E. Dokur, and R. Bayindir, "Smart coordination of predictive load balancing for residential electric vehicles based on EMD-Bayesian optimised LSTM," *IET Renewable Power Generation,* vol. 16, no. 15, pp. 3216-3232, 2022/11/01 2022, doi: https://doi.org/10.1049/rpg2.12572.

[25] E. T. Šimon Appelt, Dan Li, Georgios Tsatsaronis, "Automated Machine Learning with applications in Elsevier Topic Pages and Covid-19 Relevancy Algorithm," *SSRN,* pp. 2-3, 2021.

[26] R. B. Cleveland, W. S. Cleveland, and I. Terpenning, "STL: A Seasonal-Trend Decomposition Procedure Based on Loess," (in English), *Journal of Official Statistics,* vol. 6, no. 1, p. 3, Mar 1990

2013-01-07 1990. [Online]. Available: https://www.proquest.com/scholarly-journals/stl-seasonal-trend-decomposition-procedure-based/docview/1266805989/se-2?accountid=14723

https://resolver.library.uq.edu.au/?&genre=article&sid=ProQ:&atitle=STL%3A+A+Seasonal-Trend+Decomposition+Procedure+Based+on+Loess&title=Journal+of+Official+Statistics&issn=0282423X&date=1990-03-01&volume=6&issue=1&spage=3&author=Cleveland%2C+Robert+B%3BCleveland%2C+William+S%3BTerpenning%2C+Irma.

[27] "Statsmodels," in *Github*, ed, 2023.

[28] N. Relic. "Create smoother charts with sliding windows." https://docs.newrelic.com/docs/query-your-data/nrql-new-relic-query-language/nrql-query-tutorials/create-smoother-charts-sliding-windows/ (accessed 31-08, 2023).

[29] R. Bro and A. K. Smilde, "Principal component analysis," *Anal. Methods,* vol. 6, no. 9, pp. 2812-2831, 2014, doi: 10.1039/c3ay41907j.

[30] Raghavan. "Principal component analysis (PCA): Explained and implemented." https://medium.com/@raghavan99o/principal-component-analysis-pca-explained-and-implemented-eeab7cb73b72 (accessed 31-08, 2023).

[31] F. Murtagh and P. Contreras, "Algorithms for hierarchical clustering: an overview," *WIREs Data Mining and Knowledge Discovery,* vol. 2, no. 1, pp. 86-97, 2012, doi: 10.1002/widm.53.

[32] S. Nikbakht, C. Anitescu, and T. Rabczuk, "Optimizing the neural network hyperparameters utilizing genetic algorithm," *Journal of Zhejiang University-SCIENCE A,* vol. 22, no. 6, pp. 407-426, 2021, doi: 10.1631/jzus.a2000384.

[33] Z. Li, F. Liu, W. Yang, S. Peng, and J. Zhou, "A Survey of Convolutional Neural Networks: Analysis, Applications, and Prospects," *IEEE Transactions on Neural Networks and Learning Systems,* vol. 33, no. 12, pp. 6999-7019, 2022, doi: 10.1109/tnnls.2021.3084827.

[34] M. L. Mastery. "Bayesian Optimization- Math and Algorithm Explained." https://www.youtube.com/watch?v=ECNU4WIuhSE&ab_channel=MachineLearningMastery (accessed 26/9, 2023).

[35] S. Zhong, S. Liu, and Z. Shen, "Study on the Basic Probability Assignment Based on Grey Relational Analysis and Gaussian Membership," *IEEE Access,* vol. PP, pp. 1-1, 02/08 2021, doi: 10.1109/ACCESS.2021.3057707.
[36] P. Charoenkwan, S. W. Fang, and S. K. Wong, "A Study on Genetic Algorithm and Neural Network for Implementing Mini-Games," in *2010 International Conference on Technologies and Applications of Artificial Intelligence*, 18-20 Nov. 2010 2010, pp. 158-165, doi: 10.1109/TAAI.2010.35.
[37] A. Gjelaj and J. Balic, *Multi-objective Optimization for Milling Operation by Using Genetic Algorithm*. 2013.
[38] J. Brownlee, *Data Preparation for Machine Learning: Data Cleaning, Feature Selection, and Data Transforms in Python*. Machine Learning Mastery, 2020.
[39] SciKit-Learn. "sklearn.decomposition.PCA." https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html (accessed 15/09, 2023).
[40] SciKit-Learn. "Sklearn Github." https://github.com/scikit-learn/scikit-learn (accessed 15/09, 2023).
[41] SciKit-Optimize. "scikit-optimize github." https://github.com/scikit-optimize/scikit-optimize (accessed 15/09, 2023).
[42] keras-tuner. "keras-tuner github." https://github.com/keras-team/keras-tuner (accessed 15/09, 2023).
[43] TensorFlow. https://github.com/tensorflow/tensorflow (accessed 15/09, 2023).
[44] P. Dash. "Plotly Dash github." https://github.com/plotly/dash (accessed 15/09, 2023).
[45] D. A. Gallery. "Wind Speed Dashboard." https://dash.gallery/dash-wind-streaming/ (accessed 15/09, 2023).
[46] D. Gaul. "Thesis github." https://github.com/Gaulgeous/Energy-Forecasting (accessed 15/09, 2023).